UNIVERSITY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «Cybersecurity and Data Science»

ΠΜΣ «Κυβερνοασφάλεια και Επιστήμη Δεδομένων»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Assessing the Security Risks of Medical Mobile Applications – A comparative case study in Android and iOS platforms** Αποτίμηση Κινδύνων Ασφάλειας Κινητών Ιατρικών Εφαρμογών – Συγκριτική μελέτη περίπτωσης σε περιβάλλον Android και iOS
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**Christos – Marios Markellos** Χρήστος – Μάριος Μάρκελλος
Father's name: Πατρώνυμο:	**Michail Markellos** Μιχαήλ Μάρκελλος
Student's ID No: Αριθμός Μητρώου:	ΜΠΚΕΔ21030
Supervisor: Επιβλέπων:	**Supervisor: Panayiotis Kotzanikolaou, As. Professor** Επιβλέπων: Παναγιώτης Κοτζανικολάου, Αν. Καθηγητής

May 2023 / Μάϊος 2023

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Christos Douligeris	**Konstantinos Patsakis**	**Panayiotis Kotzanikolaou**
Professor	**As. Professor**	**As. Professor**
Χρήστος Δουληγέρης	Κωνσταντίνος Πατσάκης	Παναγιώτης Κοτζανικολάου
Καθηγητής	Αν. Καθηγητής	Αν. Καθηγητής

Abstract

In this thesis, a comprehensive security analysis of 70 popular medical mobile applications was conducted, tested in both Android and iOS platforms, for a total of 140 apps analyzed. The basic methodology includes looking for side channel leaks that can be abused by third-party applications installed on the device, assessing support for old and potentially vulnerable versions of Android and iOS, evaluating device and application integrity protections, conducting dynamic and static analysis to observe runtime behavior such as SSL usage and local data storage practices, and searching for hardcoded keys or other sensitive information embedded in the code. Also, traffic analysis is included to observe communication patterns between the mobile applications and their associated APIs. The overall findings reveal significant underlooked risks in this area. The vast majority of the apps we analyzed, lacked standard security safeguards such as SSL pinning and root detection.

The study highlights the importance of ensuring that medical apps comply with security standards and undergo rigorous testing before being made available to the public. Overall, the findings of this study underscore the need for increased attention to mobile application security, particularly in the healthcare industry where data privacy and security are of paramount importance.

Keywords: Mobile Application Security, Security Controls, Mobile Device Management, Secure Software Development Lifecycle (SDLC), Secure Communication Protocols.

Περίληψη

Στην παρούσα μεταπτυχιακή διατριβή, πραγματοποιήθηκε μια ολοκληρωμένη ανάλυση ασφαλείας 70 δημοφιλών ιατρικών εφαρμογών για κινητές συσκευές, που δοκιμάστηκαν τόσο σε πλατφόρμες Android όσο και σε iOS, για συνολικά 140 εφαρμογές που αναλύθηκαν. Η βασική μεθοδολογία περιλαμβάνει αναζήτηση διαρροών πλευρικού καναλιού που μπορούν να χρησιμοποιηθούν από εφαρμογές τρίτων που είναι εγκατεστημένες στη συσκευή, αξιολόγηση υποστήριξης για παλιές και δυνητικά ευάλωτες εκδόσεις Android και iOS, αξιολόγηση ακεραιότητας συσκευών και εφαρμογών, διεξαγωγή δυναμικής και στατικής ανάλυσης για παρατήρηση συμπεριφοράς χρόνου εκτέλεσης, όπως χρήση SSL και πρακτικές αποθήκευσης τοπικών δεδομένων, καθώς και αναζήτηση για κωδικοποιημένα κλειδιά ή άλλες ευαίσθητες πληροφορίες που είναι ενσωματωμένες στον κώδικα. Επίσης, περιλαμβάνεται ανάλυση επισκεψιμότητας για την παρατήρηση προτύπων επικοινωνίας μεταξύ των εφαρμογών για κινητά και των συσχετισμένων τους API. Τα συνολικά ευρήματα αποκαλύπτουν σημαντικούς κινδύνους που παραβλέπονται σε αυτόν τον τομέα. Η συντριπτική πλειονότητα των εφαρμογών που αναλύσαμε δεν διέθεταν τυπικές δικλείδες ασφαλείας, όπως SSL Pinning και εντοπισμός Root.

Η μελέτη υπογραμμίζει τη σημασία της διασφάλισης ότι οι ιατρικές εφαρμογές συμμορφώνονται με τα πρότυπα ασφαλείας και υποβάλλονται σε αυστηρές δοκιμές προτού διατεθούν στο κοινό. Συνολικά, τα ευρήματα αυτής της μελέτης υπογραμμίζουν την ανάγκη για αυξημένη προσοχή στην ασφάλεια των εφαρμογών για κινητά, ιδιαίτερα στον κλάδο της υγειονομικής περίθαλψης όπου το απόρρητο και η ασφάλεια των δεδομένων είναι υψίστης σημασίας.

Λέξεις-κλειδιά: Ασφάλεια εφαρμογών για φορητές συσκευές, Έλεγχοι ασφαλείας, Διαχείριση φορητών συσκευών, Κύκλος ζωής ασφαλούς ανάπτυξης λογισμικού (SDLC), Πρωτόκολλα ασφαλούς επικοινωνίας.

Thesis Contents

Figure Contents

1 Introduction

Mobile applications, have become an integral part of our daily lives. From social networking to banking, and healthcare to entertainment, mobile apps have revolutionized the way we interact with technology. However, with this convenience comes potential security risks, particularly when it comes to sensitive data.

Mobile apps are vulnerable to security risks such as data breaches, malware, and hacking attacks. Mobile devices are also vulnerable to physical theft or loss, which can result in the compromise of personal information stored on the device. Furthermore, many mobile apps collect user data, such as location information and personal contacts, which can be used for marketing or other purposes without the user's consent.

The healthcare industry is particularly vulnerable to mobile app security risks, as medical apps may collect and transmit sensitive patient data such as medical history, symptoms, and medication schedules. In order to ensure patient privacy and prevent data breaches, it is crucial to assess the security risks of medical mobile applications and implement measures to mitigate them.

1.1 Scope

The scope of this paper is to assess the security risks of medical mobile applications on both Android and iOS platforms. By identifying common security vulnerabilities and discussing potential solutions, we hope to promote safe and secure use of medical mobile applications and contribute to the development of security standards in the healthcare industry. The rise of medical mobile applications has enabled patients to manage their health more effectively, from tracking medication schedules to monitoring symptoms. However, these applications also present significant security risks, particularly when it comes to sensitive medical data. In this paper, the assessment of the security risks of 70 medical mobile applications on both Android and iOS platforms will be analyzed.

1.2 Environment

In security analysis and application development, many factors can depend on the used environments (operating system (OS), version, programs, etc.) and the available and used tools. For this reason, the following sections list the primary factors and conditions used during this thesis creation.

1.2.1 Computer and Operating System

The basic computer used was an ACER Aspire 7 from year 2019 with Windows 10 operating system running on it. The feature that made this computer reliable for the aspects of this thesis was the architecture of its Central Processing Unit (CPU), using eight total cores equipped with x64 architecture.

Another tool that was used was Virtual Box, equipped with host Kali Linux 2022.1 version operating system installed on it. In this machine was installed all the required security tools for analysis reasons.

1.2.2 Physical Phones

During this thesis creation, in the initial structure, has been used an android physical phone Samsung Galaxy with Android 5.1 Lollipop running on it.

Another physical devices used, was the basic iPhone 11 with the latest installed IOS version of 16.1 which is applicable for all app installation and a jailbroken iPhone 5s, which was identical for the jailbreak device detection analysis.

1.2.3 Emulated Devices

All the basic analysis for Android applications has been performed using emulated devices, which are listed below:

- Android 7.1.1 x86 (Google APIs)
- Android 8 x86 (Google APIs), which is rooted.

1.3 Security Tools

This thesis tries to use as many and as diverse for static and dynamic analysis, penetration testing, and reverse engineering tools as possible. For all the below listed tools a separate chapter is following in the Chapter 3 for specific technical details.

The basic tool used was **mobSF**, for android .apk and ios .ipa file analysis and reporting.

Specifically for Android apps were used the below tools:

- **ApkTool** (Specifically for decompile and compile .apk files)
- **JarSigner** (Used for sign the decompiled .smali file)
- **Frida** (For Hooking basic function or methods called)
- **JDB** (Debugger to investigate running application using process id)
- **BurpSuite** (To identify original API calls between application and server side)

1.4 Style of the Thesis

This thesis is presenting a security analysis of 70 medical health – mHealth, apps analyzed in both IOS and Android platforms (in total 140).

Regarding the structure, in first chapter an extensive analysis of mobile security landscape will be presented for specific categories such as: device-side, network-side, application-side and user side.

In the second chapter, platform security analysis will take place for both Android and IOS, analyzing also the existing security mechanisms found on them, pointing also to some critical mechanisms that used extensively to mobile operating systems. Also, mHealth validity and meaning is explained, presenting also the improvements in health management and future – technological innovations.

In addition, in third chapter, an extensive analysis will be presented for all mechanisms used for both android and iOS applications for categories:

- **Side Channel Leaks**
- **SSL Findings (SSL Pinning, SSL/TLS HTTPS Usage)**
- **Device Integrity (Root Detection)**
- **Application Protection Integrity (Apktool, JarSigner, JDB, Obfuscation)**
- **HardCoded keys**

portraying also all the relevant metrics from our significant findings in chapter 4.

Finally, in chapter 7, general improvements and techniques are proposed for future research for both preventing existing vulnerabilities and design more security applications with standard security practices.

2 Background – Relates Work

In this chapter a whole overview of not only Mobile Application Landscape and Mobile Operating Systems but also the overall background and newly technologies of Mobile mHealth Applications will be analysed per each respective category.

2.1 Mobile Security Landscape

The enterprise mobile application security threats landscape is growing larger year over year. Increased enterprise mobility, meaning allowing access to the organization's network from a remote location, results in a growing variety of unprotected endpoints vulnerable to outside attacks. To gain a better understanding of the threats landscape revolving around enterprise mobile application security, ASEE discusses the four main threat categories in today's cyber security environment.

Mobile security is to protect smartphones, tablets, laptops, and other computing devices. Mobile security is also known as wireless security. Mobile security has increased in recent years. To secure devices from thief, phishing, hacker mobile security is used. There are many organizations which have many data and data should be protected from hacker. Information should be private and should not be leaked. Company needs to protect their devices. The best way to protect company data is not to store the data in client devices. If somebody wants to know the data then they should need to get access permitted only over the network, there is no local copy to lose if a laptop or PDA is stolen or lost. This method also protects PCs in the office. This can be more convenient for a worker to work from local copy of data on a laptop transported from home or on a thumb drive, the high ability of broadband access and the maturity of remote access the technologies such as laptops and smart phones, which is much or less convenient. This approach provides better security while still letting people work in many locations and in many devices.

"The Mobile Security Landscape" by Christoffer Kønig, presents a survey of the mobile security landscape and discusses various security challenges, such as the increasing number of mobile devices and the diversity of mobile operating systems [1].

Figure 1: Mobile Security Landscape Categories.

2.1.1 Device-Side Security Threats

Enterprises nowadays have a BYOD (bring-your-own-device) policy in place. It allows the employees to use their personal mobile devices to install enterprise apps or connect them to the office network. By doing so, mobile devices automatically become a potential weak endpoint exposing the organization to external threats.

The threats personal mobile devices carry, range from insecure mobile apps, too few restrictions regarding applications accessing the data stored on the device, as well as OS vulnerabilities. A malicious app installed on the user's phone can easily harvest credentials used to connect to the organization's network. These are just some of the examples of how a mobile device can be a potential threat to an enterprise's security infrastructure. Educate your employees about the threats and consequences a cyberattack can cause.

To protect against these device-side security threats, users should always keep their device's operating system up-to-date, use strong passwords or biometric authentication, avoid downloading apps from untrusted sources, and enable device encryption if available. Additionally, users should be cautious when giving physical access to their devices and should always be aware of their device's security settings.

The device-side security threats can be divided to the below categories:

Data Leaks

Data leaks can occur in various ways, such as through malicious apps, phishing attacks, unsecured networks, and physical theft or loss of devices. Once sensitive information is leaked, it can be used for various malicious purposes such as identity theft, fraud, and cyber attacks.

According to various reports and studies, data leaks are becoming increasingly common in the mobile security landscape. For example, in 2020, the number of reported data breaches in the United States reached a record high of over 1,000, with the majority of these incidents involving mobile devices.

Some of the most common types of data leaked from mobile devices include personal identifiable information (PII) such as names, addresses, and Social Security numbers, financial information such as credit card details and bank account information, and sensitive corporate information such as trade secrets and intellectual property.

Respective analysis of privacy protection was performed by Joseph Chan Joo Keng [2] with the goal of the correlation of user actions to data leaks.

To mitigate the risk of data leaks, it is important to take measures such as using strong passwords, keeping mobile operating systems and apps up-to-date, avoiding unsecured networks, using encryption and two-factor authentication, and being cautious of suspicious emails or messages. Additionally, users can also consider using mobile security software that can help detect and prevent data leaks.

Open WiFi

Open WiFi networks are wireless networks that do not require a password to connect, and they are often found in public places such as cafes, airports, and hotels.

The security risks associated with open WiFi networks are mainly due to the fact that they are unencrypted, which means that any data transmitted over these networks is vulnerable to interception and eavesdropping by cybercriminals. This can include sensitive information such as passwords, usernames, credit card details, and other personal information. Related work, examined the users' awareness of privacy leakage in public hotspots [3].

Hackers can use various methods to intercept data transmitted over open WiFi networks, such as man-in-the-middle (MITM) attacks, where they intercept and modify data packets sent

between the user and the internet, or they can set up rogue access points that mimic legitimate networks to trick users into connecting.

Phishing attacks

Phishing is a type of cyber attack where an attacker impersonates a trusted entity, such as a bank or a social media site, to trick the victim into revealing sensitive information such as usernames, passwords, and credit card details. Can occur through various channels, including email, text messages, social media, and messaging apps. In the context of mobile devices, phishing attacks may target users through mobile-specific channels such as push notifications or mobile-specific apps [4].

Phishing attacks are designed to exploit human vulnerabilities, such as trust and urgency, to trick users into taking actions that compromise their security. For example, a phishing email may contain a link to a fake login page that looks like the legitimate site, where the user is prompted to enter their login credentials. Once the user enters their credentials, the attacker can use them to gain access to their accounts.

Mobile applications lack security properties such as confidentiality, authentication, authorization, data integrity, and non-repudiation. So, mobile apps should be less trusted than desktops [4] and the trust also depends on platform security of application permission in the environment of either Android or iOS [5].

Spyware

Spyware is a type of malicious software that is designed to secretly monitor and collect information from a device without the user's knowledge or consent. In the mobile security landscape, spyware can be a serious threat to users, as it can compromise their privacy and security. Malware and malicious software such as spyware, Trojans, and bots are used to carry out unauthorized operations on a targeted system in order to steal information or disrupt the system [6].

Spyware can be installed on a mobile device through various methods, such as through phishing attacks, app downloads, or even physical access to the device. Once installed, spyware can monitor and collect various types of information, such as call logs, text messages, browsing history, location data, and even keystrokes.

Some common types of malicious apps include those that steal user data, such as login credentials and personal information, those that send spam messages or make unauthorized charges, and those that hijack a device and use it to launch attacks on other devices or networks.

Malicious apps can be distributed through various channels, including unofficial app stores, phishing websites, and even legitimate app stores if the app is able to bypass security measures. Once installed, malicious apps can often run in the background without the user's knowledge, allowing them to perform malicious activities undetected.

Apps with weak security

Apps with weak security can pose a significant threat to users, as they can expose them to various types of cyber attacks and data breaches. Weak security in apps can manifest in various ways, such as poor encryption practices, insecure data storage, or insufficient user authentication measures [7].

If an app has weak security, it can make it easier for attackers to exploit vulnerabilities and gain unauthorized access to sensitive data or user accounts. This can include personal and financial information, as well as login credentials for other accounts associated with the user.

The risk posed by apps with weak security has become more significant with the increasing use of mobile devices for sensitive transactions, such as online banking and shopping. A successful

attack on an app with weak security can result in significant financial losses for the user, as well as reputational damage to the app's developer.

Out-of-date devices

Out-of-date devices may be more vulnerable to cyber attacks and exploits. When a device's operating system or applications are not up-to-date, known vulnerabilities may remain unpatched, leaving the device and its data at risk [8].

Attackers may exploit these vulnerabilities to gain unauthorized access to the device or steal sensitive information, such as login credentials and personal data. They may also use the device as part of a larger botnet, which can be used to carry out distributed denial-of-service (DDoS) attacks and other malicious activities.

Identity theft

Attackers may be able to steal personal information, such as Social Security numbers and credit card information, and use it to carry out fraudulent activities. Can occur through various methods, such as phishing attacks, malware-infected apps, and social engineering. Attackers may use stolen information to open new accounts in the victim's name, make unauthorized purchases, or even apply for loans and credit cards.

Mobile devices are particularly vulnerable to identity theft, as they often contain a large amount of personal information and are frequently used to access sensitive accounts, such as banking and financial accounts. If a mobile device is lost or stolen, this can also increase the risk of identity theft.

"Mobile Apps and Identity Theft: An Exploratory Study" by Harsh Verma, examines the potential risks of identity theft associated with the use of mobile apps, including vulnerabilities in app security, data breaches, and phishing attacks [9].

Operating system vulnerabilities

The operating system (OS) of a mobile device is a critical component that can significantly impact the overall security of the device. The OS is responsible for managing various functions of the device, including access to data, hardware, and applications, and can be vulnerable to a wide range of security threats.

Mobile OS security threats can include malware infections, remote exploits, and vulnerabilities in the code of the OS itself. Malware can compromise the device's security by stealing sensitive data, tracking the user's location and activities, and even taking control of the device. Remote exploits can allow attackers to gain access to the device without the user's knowledge or consent, while vulnerabilities in the code of the OS can leave the device open to various types of attacks.

"An Analysis of Operating System Vulnerabilities in Mobile Devices" by Tuba Yavuz, provides an analysis of operating system vulnerabilities in mobile devices, including the frequency and severity of such vulnerabilities and their impact on mobile app security [10].

Physical Access

When a device falls into the wrong hands, an attacker can potentially gain access to sensitive information, such as login credentials, personal data, and even financial information. Can occur through various means, such as theft or loss of the device, as well as unauthorized access by someone with legitimate access to the device. This can include family members, friends, or colleagues who may be able to access the device when it is left unattended.

"Investigating the Security of Mobile Health Applications" by Karthikeyan Shanmugam, investigates the security of mobile health applications and examines the impact of physical access to devices on the confidentiality, integrity, and availability of sensitive health data [11].

Unauthorized access to device settings

Access to device settings can allow an attacker to make changes to the device's security settings or install malicious applications without the user's knowledge. Can occur through various means, such as social engineering attacks or the use of vulnerabilities in the device's operating system. Social engineering attacks may involve tricking the user into providing their login credentials or other sensitive information, while vulnerabilities in the operating system can allow an attacker to bypass security measures and gain access to device settings.

"Mobile Application Security: A Survey" by Muazzam A. Khan and Muhammad Ali Babar, provides a survey of mobile application security and discusses various security issues related to unauthorized access to device settings [12].

Side-loading of apps

Sideloading is the process of installing an application onto a device from a source other than the official app store. Can introduce security risks to a device, as applications downloaded from unofficial sources may not have undergone the same level of scrutiny and testing as those available on the official app store. This can lead to the installation of malicious applications or apps that have vulnerabilities that can be exploited by attackers [26].

Additionally, side - loading may require users to grant permissions to the app that they may not fully understand. This can allow the app to access sensitive data or device settings, which can be exploited by attackers.

"A Survey of Android Security Threats and Defenses" by Xiaoyin Wang, et al, provides a survey of Android security threats and defenses, including side - loading apps and countermeasures to mitigate these threats [13].

2.1.2 Application-Side Security Threats

Mobile security applications have been developed to address these risks, but they too are not immune to side threats.

Jailbroken or rooted devices, which have been modified to allow users to bypass restrictions imposed by the manufacturer or service provider, are one such threat. These devices may allow malicious actors to gain access to sensitive information or take control of the device.

Reverse engineering is another threat that mobile security applications face. This process involves analyzing and deconstructing the software to understand its inner workings, potentially exposing vulnerabilities and allowing attackers to exploit them.

Hooking and debuggers are other techniques used by attackers to intercept data and manipulate applications. They may be used to inject malicious code or to modify the behavior of the application.

Poor encryption practices can also leave mobile security applications vulnerable to attack. If encryption keys are not properly secured or if the encryption algorithm is weak, attackers may be able to decrypt sensitive information.

JailBroken/Rooted Devices

Jailbroken or rooted devices are mobile devices that have undergone unauthorized modifications to bypass security protocols and gain root access to the operating system [14].

"Protecting Sensitive Data in Untrusted Android Environments Using Trusted Execution Environments" by Xiaojing Liao, presents a method for protecting sensitive data in untrusted Android environments, including the detection of jailbroken/rooted devices [15].

"Detecting Jailbroken iOS Devices" by Julian Schütte and Eric Bodden, presents a method for detecting jailbroken iOS devices, including the detection of jailbreak detection mechanisms used by mobile apps.

These devices pose a significant security threat to application developers and end-users. Here are some of the application side security threats for jailbroken/rooted devices that you can consider for the current thesis:

1. **Application Tampering**: Jailbroken/rooted devices have elevated privileges, which allow users to modify the behavior of applications. Hackers can use this to modify an application's code, inject malicious code, or alter the behavior of the application, leading to security breaches.
2. **Data Theft**: Jailbroken/rooted devices can allow unauthorized access to the device's file system, enabling hackers to steal sensitive information such as login credentials, credit card information, and personal data.
3. **Malware Attacks**: Hackers can distribute malware through third-party app stores, which are commonly used on jailbroken/rooted devices. Malware can take control of the device, steal data, or use the device for malicious activities.
4. **Reverse Engineering**: Hackers can use jailbroken/rooted devices to reverse engineer applications to discover vulnerabilities that can be exploited to launch attacks against other devices or networks.
5. **App Store Abuse**: Jailbroken/rooted devices can bypass app store restrictions, allowing users to download and install untrusted applications. These applications may contain malware or be used to launch attacks on other devices.
6. **Debugging**: Debugging tools can be used on jailbroken/rooted devices to gain insights into the application's behavior, including sensitive data such as encryption keys.

Reverse Engineering

Reverse engineering is the process of analyzing a product or device to understand its design and functionality.

"A Survey on Reverse Engineering of Android Applications" by Hui Wang and Xinming Ou, provides a comprehensive survey on the reverse engineering of Android applications, including various reverse engineering techniques and tools [16].

In the context of mobile apps, reverse engineering is often used to analyze the code and resources of an app to understand how it works and potentially extract sensitive information.

There are various tools and techniques that can be used for reverse engineering mobile apps. Some of the commonly used tools include:

1. **Decompilers**: These tools can be used to decompile the compiled code of an app, which can help in understanding how the app works and potentially identifying any vulnerabilities or weaknesses.
2. **Debuggers**: Debuggers can be used to step through the code of an app and identify any bugs or vulnerabilities that may exist.
3. **APK Extractors**: These tools can be used to extract the APK (Android Application Package) file of an app, which can then be analyzed using other tools.

4. **Obfuscation**: Some developers use obfuscation techniques to make it harder to reverse engineer their apps. Obfuscation tools can be used to reverse the effects of these techniques and gain a better understanding of the app's code.

Reverse engineering can be used for legitimate purposes, such as analyzing the code of an app to identify security vulnerabilities and improve its security. However, it can also be used for malicious purposes, such as stealing intellectual property or personal data. Therefore, it is important to use reverse engineering tools and techniques responsibly and ethically.

Hooking Attacks

Hooking attacks in mobile apps involve an attacker intercepting the flow of code execution in an app and modifying it to their advantage.

"Hooking Detection Techniques for Mobile Applications" by Timothy D. Morgan, presents a survey of hooking techniques used in mobile application security, along with detection techniques to identify and prevent hooking attacks [17].

"A Survey of Hooking Techniques and Their Applications" by Jiong Zhang and Xiaolin Gui, provides an overview of hooking techniques used in various applications, including mobile apps, and analyzes their security implications [18].

The attacker can use these attacks to bypass security measures, steal sensitive data, or perform other malicious actions.

There are several types of hooking attacks that can be carried out on mobile apps, including:

1. **Function Hooking**: In this type of attack, an attacker modifies the function calls in an app to redirect the flow of code execution to a malicious function.
2. **Method Swizzling**: Method swizzling is a technique used by iOS developers to dynamically replace the implementation of a method at runtime. Attackers can use this technique to replace legitimate code with malicious code.
3. **Inline Hooking**: In this type of attack, an attacker modifies the instructions of an app's code to redirect it to a malicious function.
4. **Memory Hooking**: Memory hooking involves modifying the memory of an app to redirect its execution to a malicious function.

To carry out a hooking attack, an attacker needs to have access to the code of the app, which can be obtained through reverse engineering techniques. Once the code has been analyzed, the attacker can identify the functions and methods that can be hooked and modified.

To protect against hooking attacks, developers can implement security measures such as code obfuscation, anti-debugging techniques, and integrity checks. They can also use runtime protection mechanisms such as binary code signing and encryption to prevent attackers from tampering with the app's code.

Debuggers

There are several tools examples of tools used for debugger-based attacks in mobile apps: Debuggers are software tools used by developers to analyze and debug the code of an application. However, they can also be used by attackers to steal sensitive information from mobile apps.

Attackers can use debuggers to modify the execution of an app's code to their advantage, intercept and inspect network traffic, and access sensitive data such as usernames, passwords, and other personal information. They can also use debuggers to bypass security measures implemented by the app.

"Debugger Detection Techniques for Android Applications" by Samaneh Tajalizadehkhoob, proposes a set of techniques to detect the use of debuggers in Android applications, which can be used to prevent or mitigate attacks that rely on debuggers [19].

To carry out a debugger-based attack, the attacker needs to have physical access to the device or use other techniques such as jailbreaking or rooting to gain access to the app's code. Once the code has been accessed, the attacker can use a debugger to analyze the app's behavior, identify vulnerabilities, and steal sensitive information.

1. **Frida**: Frida is a dynamic instrumentation tool that allows attackers to inject their own code into the runtime of an application, modify its behavior, and steal sensitive information.
2. **Xposed**: Xposed is a framework that allows attackers to modify the behavior of an app by intercepting method calls and replacing them with their own code.
3. **JDB**: JDB is a powerful debugger that can be used to inspect and modify the code of an app.
4. **IDA Pro**: IDA Pro is a disassembler and debugger that can be used to analyze and reverse-engineer the code of an app.
5. **Hopper**: Hopper is a reverse engineering tool that can be used to disassemble, decompile, and debug the code of an app.

These tools can be used by attackers to steal sensitive information such as passwords, banking information, and personal data.

To protect against debugger-based attacks, developers can implement security measures such as code obfuscation, anti-debugging techniques, and runtime protection mechanisms such as binary code signing and encryption. Additionally, developers can implement security best practices such as secure coding, data encryption, and secure communication protocols to minimize the risk of data theft.

Screen Recording

Screen recording mobile apps can be used by attackers to record and steal sensitive information such as login credentials, personal information, and banking information.

"Detecting Screen Capture-based Attacks on Android Devices Using Motion Sensors" by Alireza Sadeghi, proposes a technique for using motion sensors to detect screen capture-based attacks on Android devices [20].

These apps allow the attacker to record the user's screen and capture any information that is displayed on it.

Here are some examples of screen recording mobile apps that can be used for malicious purposes:

1. **AZ Screen Recorder**: AZ Screen Recorder is a popular screen recording app that can be used to record the screen of an Android device.
2. **DU Recorder**: DU Recorder is another screen recording app that can be used to capture the screen of an Android device.
3. **iOS Screen Recorder**: iOS Screen Recorder is an app that can be used to record the screen of an iPhone or iPad.
4. **AirShou**: AirShou is a screen recording app that can be used to capture the screen of an iOS device.
5. **Mobizen**: Mobizen is a screen recording app that can be used to capture the screen of an Android device.

Attackers can trick users into downloading these apps by disguising them as legitimate apps or by using social engineering techniques such as phishing attacks. Once the app is installed

on the user's device, the attacker can use it to record the user's screen and capture sensitive information.

Poor Encryption/Storage

Mobile apps can store data insecurely on the device itself or in cloud storage. If the data is not properly encrypted or protected, it can be easily accessed by unauthorized parties.

"Security Analysis of iOS Password Vaults" by Andi Bejleri, examines the security of password vaults in iOS apps, and identifies a number of vulnerabilities related to poor encryption and storage practices [21].

"Encryption in Android Applications: A Comparative Study" by Sami Qasem, compares the encryption practices of different Android apps, and identifies a number of weaknesses and vulnerabilities [22].

Mobile apps often store sensitive data such as user credentials, personal information, and financial data. If this data is not properly encrypted and stored, it can be vulnerable to theft and misuse by attackers.

Some examples of poor encryption/storage practices that can make mobile apps vulnerable to attack are categorized below:

1. **Storing data in plain text**: If sensitive data is stored in plain text, it can be easily accessed by attackers who gain access to the app's data storage.
2. **Using weak encryption**: If the app uses weak encryption algorithms or keys, attackers can easily crack the encryption and access the sensitive data.
3. **Storing encryption keys in the app's code**: If the app's encryption keys are stored in the app's code, they can be easily extracted by attackers who reverse engineer the app.
4. **Failing to implement secure data transfer protocols**: If the app fails to implement secure communication protocols such as SSL/TLS, attackers can intercept and steal data transmitted over the network.

To protect against poor encryption/storage practices, developers should implement strong encryption algorithms and keys, and store encryption keys securely, such as using hardware security modules (HSMs) or key management services (KMS). They should also use secure communication protocols such as SSL/TLS to protect data during transmission.

Insecure Application Code

Insecure application code is a major vulnerability for mobile apps. If the app's code contains security flaws or vulnerabilities, it can be exploited by attackers to steal sensitive data or gain unauthorized access to the device.

"Analyzing the Security of Third-Party iOS Applications" by Matthew Smith and Sam Small, investigates the security of third-party iOS apps, and finds that many of these apps contain insecure coding practices that can lead to vulnerabilities [23].

Here are some examples of insecure application code practices that can make mobile apps vulnerable to attack:

1. **Input validation**: If the app fails to properly validate user input, it can be vulnerable to attacks such as SQL injection and cross-site scripting (XSS).
2. **Insecure data storage**: If the app stores sensitive data in insecure locations or fails to properly encrypt the data, it can be vulnerable to theft and misuse by attackers.

3. **Authentication and authorization**: If the app fails to properly authenticate users or authorize access to sensitive data, it can be vulnerable to attacks such as brute force attacks and session hijacking.
4. **Lack of server-side controls**: If the app relies solely on client-side controls, it can be vulnerable to attacks such as tampering and reverse engineering.

To protect against insecure application code, developers should follow secure coding practices and regularly perform security testing to identify and address vulnerabilities in the app's code. They should also implement security measures such as input validation, encryption, and server-side controls to protect sensitive data and prevent unauthorized access.

Insecure Communication

Insecure communication is a significant vulnerability for mobile apps. If the app fails to implement secure communication protocols, such as SSL/TLS, it can be vulnerable to attacks such as eavesdropping, man-in-the-middle attacks, and data interception.

"Analysis of Insecure Communication Channels in Mobile Applications" by Shubham Jain and Pramod Kumar. This paper analyzes the insecure communication channels used in mobile apps and identifies potential security vulnerabilities [24].

Here are some examples of insecure communication practices that can make mobile apps vulnerable to attack:

1. **Failure to implement secure communication protocols**: If the app fails to use secure communication protocols such as SSL/TLS, attackers can intercept and steal data transmitted over the network.
2. **Insecure Wi-Fi connections**: If the app connects to insecure Wi-Fi networks, attackers can intercept and steal data transmitted over the network.
3. **Insufficient encryption**: If the app uses weak encryption algorithms or keys, attackers can easily crack the encryption and access the sensitive data.
4. **Unencrypted data transmission**: If the app transmits sensitive data in plain text, attackers can easily intercept and steal the data.

For protecting purposes, developers should implement secure communication protocols such as SSL/TLS to protect data during transmission. They should also use strong encryption algorithms and keys, and avoid transmitting sensitive data in plain text.

Social Engineering

Attackers can use social engineering tactics, such as phishing and other forms of deception, to trick users into installing malicious apps or providing sensitive information.

Social engineering tools are not specific to hacking mobile apps but rather are designed to exploit human vulnerabilities to gain access to sensitive information or systems.

"Mobile Application Security and Social Engineering Attacks: A Review" by S. A. E. Alshehri, reviews the security challenges in mobile app development and the different types of social engineering attacks on mobile apps [25].

Social engineering tools and techniques that attackers might use to hack mobile apps:

1. **Phishing**: Attackers may use phishing emails or text messages that appear to be from a legitimate source to trick users into divulging sensitive information such as login credentials.
2. **Pretexting**: Attackers may impersonate a trustworthy person or organization, such as a bank or IT support staff, to gain access to sensitive information.
3. **Baiting**: Attackers may leave physical devices or digital files containing malware in a public place, in the hope that someone will take the bait and open the file or plug in the device.

4. **Tailgating**: Attackers may follow an authorized person into a restricted area or access a secure system by posing as an authorized user.

Protecting against social engineering attacks, users should be cautious when responding to unsolicited emails, text messages, or phone calls that request sensitive information. They should also be wary of opening attachments or clicking on links in emails or text messages from unknown sources. In addition, users should be cautious when using public Wi-Fi networks and avoid downloading apps from untrusted sources.

2.1.3 Network-Side Security Threats

Employees are no strangers to connecting their mobile devices to public Wi-Fi. Without an added security layer, the mobile device is left vulnerable to man-in-the-middle (MitM) attacks. Combined with weak or no end-to-end encryption at all, the data stored on the mobile device could be easily targeted by a hacker. Fake Wi-Fi set up by hackers, also known as network spoofing, is a great way of baiting unsuspecting users to connect to free Wi-Fi. What follows is the user submitting their login credentials for a particular service, and the hacker in charge of the fake Wi-Fi is in possession of sensitive data.

Mobile network-side security threats refer to vulnerabilities or risks that affect the cellular network infrastructure and communication channels between mobile devices and network servers. Here are some common mobile network-side security threats:

Man-in-the-middle attacks

This type of attack involves a hacker intercepting the communication between two devices and inserting themselves into the conversation. This can allow the attacker to steal data or modify the data being sent.

"A Survey on Mobile Man-in-the-Middle Attacks" by S. Wang et al, presents a comprehensive survey on mobile Man-in-the-Middle (MitM) attacks, including attack vectors, detection techniques, and countermeasures [27].

Denial of Service (DoS) attacks

A DoS attack is a type of attack that floods a network with traffic, making it unusable for legitimate users. These attacks can disrupt mobile network services, making it difficult for users to make calls, send messages or access the internet.

"Detecting and Mitigating Flooding Attacks on Mobile Devices" by A. Alshehri, proposes a mechanism for detecting and mitigating flooding attacks on mobile devices [28].

Malware Infections

Malware can infect mobile network infrastructure and servers, allowing attackers to gain access to sensitive data or manipulate the network for their own purposes.

"A Study on the Prevalence of Malware in Android Applications" by A. Alzahrani, presents a study on the prevalence of malware in Android applications and identifies the different types of malware found in the applications [29].

Authentication and encryption weaknesses

Mobile networks rely on authentication and encryption to protect communication between devices and the network. However, weaknesses in these mechanisms can be exploited by attackers to gain access to sensitive data.

"Security Vulnerabilities in Mobile Communication Protocols" by M. M. Islam et al, explores the security vulnerabilities in mobile communication protocols, including the authentication and encryption mechanisms used in these protocols [30].

Network Spoofing

Network spoofing is another type of mobile network-side security threat. It involves an attacker impersonating a legitimate network to intercept or manipulate communication between devices and the network.

"Security Analysis of Mobile Applications and Mitigation Strategies Against Network Spoofing Attacks" by A. Khelifi, analyzes the security of mobile applications and proposes mitigation strategies against network spoofing attacks, including man-in-the-middle attacks [31].

2.1.4 User-Side Security Threats

Mobile user-side security threats refer to security risks that originate from user behavior or actions, rather than from the mobile device or app itself. Poor password security is a common user-side security threat, as users often choose easily guessable or reused passwords that can be easily cracked. Unsecured Wi-Fi networks also pose a risk, as they can be easily intercepted by attackers to steal sensitive information. Outdated software is another common threat, as older software versions may have known vulnerabilities that can be exploited by attackers to gain access to the device or data.

Poor password security

Poor password security can make it easier for attackers to gain unauthorized access to their accounts and sensitive information. Poor password security can manifest in various ways, such as using weak passwords, reusing passwords across multiple accounts, and not changing passwords frequently enough.

"Weak Password Analysis of Mobile Users" by J. Alnabulsi, S. Khan, and I. A. Khan, presented at the 2020 IEEE International Conference on Innovations in Information Technology (IIT), examines the issue of weak password usage among mobile users. The authors conducted a survey of 206 mobile users to analyze the characteristics of their passwords, and found that a significant proportion of users chose weak passwords that are easily guessable or crackable. The paper also provides recommendations for improving password security on mobile devices [32].

If a user's password is compromised, an attacker may be able to gain access to their accounts, steal sensitive data, and carry out fraudulent activities. This can include financial transactions, as well as accessing personal and confidential information.

Unsecured Wi-Fi Networks

Public Wi-Fi networks are often unsecured, which can make it easy for hackers to intercept user traffic and steal sensitive data. Attackers can set up fake Wi-Fi networks to lure users into connecting, or they can use specialized tools to eavesdrop on traffic on legitimate networks.

Goyal, S., & Singh, G, discusses the risks associated with using mobile applications on unsecured Wi-Fi networks and provides recommendations for users to stay safe while using such networks. It also proposes a framework for testing the security of mobile applications on Wi-Fi networks [33].

Outdated Software

Outdated software can contain security vulnerabilities that can be exploited by attackers to gain access to user data or control over the device.

"A Study on the Risk of Using Outdated Applications in Mobile Devices" by S. Ryu, Y. Jang, and H. J. Kim, presented at the 2017 4th International Conference on Information Science and Control Engineering (ICISCE), focuses on the risks associated with using outdated mobile applications. The authors analyze the impact of outdated applications on mobile device security and propose a system that identifies and mitigates such risks. The paper concludes that the use of outdated applications on mobile devices can pose a significant security threat, and recommends that users keep their apps updated to minimize such risks [34].

2.2 Mobile Operating Systems

Mobile operating systems (OS) are software platforms that provide the necessary environment for running mobile applications and managing hardware resources on mobile devices such as smartphones and tablets. They serve as the intermediary between the hardware and software layers, enabling users to interact with their devices and access various services and applications.

Android is an open-source mobile operating system developed by Google, which is designed to run on a variety of mobile devices such as smartphones, tablets, and smartwatches. It has a large user base and a wide range of app offerings on the Google Play Store. Android is known for its flexibility and customization options, but also for its fragmentation due to the diverse hardware and software configurations across different devices.

iOS, on the other hand, is a proprietary mobile operating system developed by Apple for its iPhone, iPad, and iPod Touch devices. It is known for its user-friendly interface and seamless integration with other Apple devices and services. iOS has a curated App Store with strict guidelines for app approval, which helps maintain its reputation for security and reliability. However, iOS is less customizable than Android due to its closed system architecture.

2.2.1 Android Operating System

Android is a popular operating system developed by Google for mobile devices, such as smartphones, tablets, and smartwatches. Android is based on the Linux kernel and uses a modified version of the Java programming language. It was first released in 2008 and has since become one of the most widely used mobile operating systems, powering more than 2 billion active devices worldwide.

Android offers a customizable and flexible user interface, with a variety of pre-installed apps and the ability to download additional apps from the Google Play Store. Android also provides support for multi-tasking, allowing users to run multiple apps at the same time.

One of the key features of Android is its open-source nature, which allows developers to customize and modify the operating system to suit their needs. This has led to a large community of developers creating custom ROMs and modifications to the Android operating system.

However, the open-source nature of Android also presents some security risks, as it can be more vulnerable to malware and other types of attacks. To address these risks, Google provides regular security updates to the Android operating system, as well as a suite of security features, such as app sandboxing, permission controls, and secure boot.

Overall, Android is a powerful and flexible operating system that offers a wide range of features and customizability for users, but it also requires vigilance to protect against security risks.

Android provides several security features to protect users and their devices from threats. Some of these security features are:

Google Play Protect

Google Play Protect developed by Google for Android devices. It is built into the Google Play Store app and is designed to continuously scan apps that are installed on a device to detect potential security threats such as malware, spyware, and harmful apps. It uses machine learning algorithms and data analytics to detect suspicious behavior and patterns in apps and on-device activity [35].

As per Google, Play Protect is enabled by default on all Android devices running Google Play Services 11 or higher. Users can access the Play Protect feature by going to the Google Play Store app and clicking on the three-line menu icon in the top left corner, then selecting "Play Protect" from the drop-down menu. The feature provides users with information on the status of their device's security, including the last time it was scanned and any detected security threats.

App Sandboxing

App sandboxing used in mobile operating systems, including Android, that limits the permissions and resources available to an app. The app is confined to a sandbox, which is a separate and isolated environment where it can only access the resources that it has been granted permission to access. This helps to prevent malicious apps from accessing sensitive information or causing damage to the device [36].

In Android, app sandboxing is enforced through a combination of kernel-level security features and the Android permission system. The Android permission system allows users to control which resources an app can access, such as the camera or microphone, while the kernel-level security features prevent apps from accessing resources outside of their designated sandbox.

Permissions System

The Android operating system provides a permission system that allows users to control the access that apps have to sensitive data and device features. This permission system serves as an important security feature in Android apps, as it ensures that apps are not granted access to sensitive data or device features without the user's consent [37].

When an app requests permission to access a sensitive data or device feature, such as the camera or location, the Android system presents a permission prompt to the user. The user can then decide whether or not to grant the permission request. If the user denies the permission request, the app is not granted access to the requested data or feature.

The permission system in Android also provides users with the ability to view and manage the permissions that have been granted to each installed app. Users can revoke permissions at any time if they no longer trust an app or want to limit its access to sensitive data or features. It is important for app developers to understand how the permission system works and to use it appropriately in their apps to ensure that user data and device features are not compromised.

Verified Boot

Verified Boot ensures the integrity of the operating system and other software components during the boot process. It works by verifying the digital signature of each component before allowing it to execute. If the signature is invalid or does not match the expected value, the boot process is halted, and the user is notified that their device may be compromised [38].

Verified Boot is implemented in the Android operating system using a technology called dm-verity. This technology uses cryptographic hashes to verify the integrity of each file in the system partition, including the kernel, the device tree, and the system image. When the device boots up, the bootloader checks the integrity of each file by verifying its hash against a known value stored in the device's read-only memory (ROM). If the hashes do not match, the device refuses to boot up, indicating that the system has been tampered with [39].

In addition, it protects against a class of attacks known as bootkits. Bootkits are malicious software that infects the bootloader, allowing attackers to control the device even before the operating system has loaded. By verifying the integrity of the bootloader and other critical components, Verified Boot makes it much more difficult for attackers to execute these types of attacks.

Encryption

Encryption protects sensitive data stored on the device from unauthorized access. Android provides built-in support for both full-disk encryption and file-based encryption [40].

Full-disk encryption (FDE) protects the entire device by encrypting all user data on the device's internal storage. When the device is locked, the encryption keys are discarded, making it virtually impossible for anyone to access the data without the correct credentials. Android devices running version 6.0 (Marshmallow) and later are required to implement FDE by default.

File-based encryption (FBE) is a more flexible form of encryption that allows individual files to be encrypted using different keys. This makes it possible to encrypt only sensitive data while leaving other data unencrypted for faster access. Android devices running version 7.0 (Nougat) and later are required to implement FBE by default.

Android also provides a framework for developers to implement encryption in their apps using the Android Keystore system. The Keystore system provides a secure location for storing encryption keys and other sensitive data, making it more difficult for attackers to access them.

Google Play app review process

The Google Play app review process helps ensure the safety and security of apps available on the Google Play Store. The review process is designed to identify and remove apps that violate Google's policies or could potentially harm users' devices or data [41].

The review process includes both automated and manual checks. Automated checks use machine learning algorithms to scan apps for common security issues, such as malware, phishing, and deceptive behavior. Manual checks are performed by human reviewers who examine apps in more detail, looking for issues such as inappropriate content or behavior.

In addition to the initial review process, Google also monitors apps on an ongoing basis to ensure they continue to meet its policies and standards. Apps that are found to violate Google's policies are removed from the Play Store, and developers may be subject to further action, such as account suspension or termination.

By implementing a thorough app review process, Google helps ensure that the apps available on the Play Store are safe and secure for users to download and use.

Security Updates

Security updates can protect devices from known vulnerabilities and exploits. Android devices receive security updates from the device manufacturer or carrier, and Google provides monthly security patches for its own devices, as well as regular security updates for the Android operating system itself [42].

Security updates typically address known vulnerabilities in the operating system or other software components, such as the browser or messaging app. These vulnerabilities can be exploited by attackers to gain access to the device or steal data, so it is essential that devices are kept up to date with the latest security patches.

In addition to providing security updates, Google also works closely with device manufacturers and carriers to ensure that devices receive timely updates. This can be a challenge,

as many devices are sold with custom versions of Android that may require additional testing and development before they can be updated [43].

To help address this challenge, Google has introduced Project Treble, a major restructuring of the Android operating system that makes it easier for device manufacturers to provide updates. Project Treble separates the device-specific code from the core Android operating system, making it easier for manufacturers to update devices without having to make significant changes to the underlying code.

2.2.2 IOS Operating Systems

iOS is the mobile operating system developed by Apple for their iPhones, iPads, and iPod Touch devices. It was first released in 2007 and has since become one of the most popular mobile operating systems, known for its user-friendly interface, security, and privacy features.

One of the key features of iOS is its closed-source nature, which means that only Apple can modify and update the operating system. This allows Apple to have greater control over the security and stability of the system, as well as the quality of the apps available on the App Store.

iOS also offers a wide range of features, including Siri, a voice-controlled personal assistant, and the ability to use Apple Pay for mobile payments. Additionally, iOS provides support for multi-tasking, allowing users to switch between apps quickly and easily.

In terms of security, iOS is known for its strict app review process, which ensures that all apps on the App Store meet Apple's guidelines and do not contain malware or other malicious code. Additionally, iOS provides a range of security features, such as Touch ID and Face ID biometric authentication, app sandboxing, and secure boot.

Overall, iOS is a powerful and user-friendly mobile operating system that offers a wide range of features and strong security and privacy protections. However, because it is a closed-source system, it can be less flexible and customizable than some other mobile operating systems, such as Android.

App Store Review Process

The App Store Review Process ensures the safety and security of apps available on the Apple App Store. The review process is designed to identify and remove apps that violate Apple's policies or could potentially harm users' devices or data [44].

The review process includes both automated and manual checks. Automated checks use algorithms to scan apps for common security issues, such as malware, phishing, and deceptive behavior. Manual checks are performed by human reviewers who examine apps in more detail, looking for issues such as inappropriate content or behavior.

In addition to the initial review process, Apple also monitors apps on an ongoing basis to ensure they continue to meet its policies and standards. Apps that are found to violate Apple's policies are removed from the App Store, and developers may be subject to further action, such as account suspension or termination.

By implementing a thorough app review process, Apple ensure that the apps available on the App Store are safe and secure for users to download and use.

Touch ID and Face ID

Touch ID and Face ID are security features in iOS that use biometric authentication to protect user data and sensitive information stored on the device. Touch ID uses a fingerprint scanner embedded in the device's Home button, while Face ID uses facial recognition technology to authenticate users [45].

These technologies allow users to unlock their devices or authenticate within apps without having to enter a password or passcode manually, making it more convenient and faster to access their devices and apps. Additionally, biometric authentication can be more secure than traditional passwords or passcodes since they are unique to each individual and more difficult to replicate or guess.

To ensure user privacy and security, Apple has implemented several measures to protect biometric data. Biometric data is stored securely on the device's secure enclave, a special hardware component separate from the device's main processor that is designed to store sensitive information. The biometric data is never shared with Apple or other third parties and is only used to authenticate the user on the device [46].

Secure Enclave

The Secure Enclave consists of a secure environment for the storage and processing of sensitive information, such as biometric data (e.g. fingerprints for Touch ID, face data for Face ID), cryptographic keys, and other authentication-related information.

The Secure Enclave is a separate hardware component embedded within the device's processor, with its own secure boot process and isolated memory space. This ensures that sensitive information stored in the Secure Enclave is protected from potential security threats, such as malware or unauthorized access attempts [47].

For the fact of protecting sensitive information, the Secure Enclave is also responsible for performing secure operations, such as cryptographic operations for encryption and decryption. This ensures that sensitive data is only accessed by authorized users or processes and that cryptographic operations are performed securely.

Data Protection

Data protection is equivalent to the encrypted data stored on the device. This includes data such as user passwords, application data, and other sensitive information. When data protection is enabled, iOS uses a combination of hardware and software encryption to protect data stored on the device. This encryption is based on a unique device-specific encryption key that is generated when the device is first set up. The encryption key is protected by the device's secure enclave, ensuring that it cannot be accessed or extracted by unauthorized parties [47].

iOS also provides additional protections to prevent unauthorized access to data. This includes measures such as requiring a passcode or biometric authentication to unlock the device, and limiting the amount of time the device can be idle before requiring reauthentication.

Permissions System

The permissions system allows users to control what data and features an app can access on their device. This helps protect user privacy and prevent apps from accessing sensitive information without the user's knowledge or consent [47].

When an app requests access to a particular feature or data, such as the device's camera or microphone, iOS prompts the user to grant or deny the request. Users can also view and manage app permissions at any time by going to the device's Settings app and selecting the app in question.

In addition includes other security features to prevent unauthorized access to data. For example, apps are sandboxed, which means that each app is isolated from other apps and the underlying operating system. This helps prevent malicious apps from accessing data or features that they should not have access to.

Two-Factor Authentication

Two-Factor Authentication (2FA) adds an additional layer of protection to user accounts by requiring users to provide a second form of authentication in addition to a password or passcode. This helps prevent unauthorized access to user accounts, even if a password or passcode has been compromised [47].

In iOS, 2FA is typically implemented using a trusted device, such as an iPhone or iPad, that is associated with the user's account. When 2FA is enabled, the user is required to enter a verification code in addition to their password or passcode when logging into their account from a new device or browser. The verification code is typically sent to the user's trusted device via text message, push notification, or phone call.

In general includes other security features to protect user accounts. For example, iOS includes a feature called "Sign in with Apple" that allows users to create and sign in to accounts using their Apple ID. This feature includes additional privacy protections, such as the ability to create a unique email address for each app or service that forwards to the user's real email address, to help protect user privacy and prevent spam.

Automatic Updates

Automatic updates ensure that users are always running the latest version of the operating system and apps on their device. This helps protect users from security vulnerabilities and other security threats that may be present in older versions of software [47].

In iOS, users have the option to enable automatic updates for both the operating system and apps. When automatic updates are enabled, iOS will automatically download and install updates in the background, without requiring any action from the user.

Additionally, includes other security features to protect users from security threats. For example, iOS includes a feature called "App Transport Security" that requires apps to use secure HTTPS connections when communicating with servers over the internet. This helps prevent attackers from intercepting and modifying data transmitted between the app and the server.

2.3 Medical Mobile Application Significance

Mobile medical applications, also known as "health apps", are software applications designed to help people manage their health and wellness. These apps can be used to track health metrics, manage medication schedules, communicate with healthcare providers, and more. The significance of mobile medical applications lies in their ability to make healthcare more accessible and convenient for people. By using these apps, patients can easily track their health data and communicate with healthcare professionals, which can lead to more personalized and effective care.

Mobile medical apps can also be used to address healthcare disparities by reaching people in remote or underserved areas who may not have access to traditional healthcare services. Additionally, mobile medical apps can help reduce healthcare costs by providing preventative care and early intervention, which can ultimately lead to fewer hospitalizations and better health outcomes.

2.3.1 Mobile Health – mHealth meaning

Mobile health, or mHealth, refers to the use of mobile devices, such as smartphones, tablets, and wearable devices, to support healthcare and medical services. This includes a wide range of applications and services that are designed to improve health outcomes, enhance patient care, and provide healthcare professionals with better tools and resources.

mHealth technologies can be used for a variety of purposes, including remote monitoring of patients, medication adherence tracking, access to health information and resources, and communication between patients and healthcare providers. mHealth also enables the use of mobile apps and other digital tools for health and wellness, such as fitness tracking, diet and nutrition tracking, and mental health support (Figure 2).

Overall, mHealth is an important area of healthcare innovation that has the potential to improve access to care, reduce costs, and enhance the quality of care for patients. As mobile devices continue to become more ubiquitous and powerful, the potential for mHealth to transform healthcare will only continue to grow.

Figure 2: Medical Mobile Applications usages.

Health and wellness apps

These apps are designed to help individuals manage their health and wellness, such as tracking their fitness goals, monitoring their diet and nutrition, and managing their mental health. These apps may also provide access to educational resources and support communities [48],[49],[50],[51].

Clinical and diagnostic apps

The design of the apps is to support clinical and diagnostic services, such as remote patient monitoring, telemedicine, and mobile diagnostic tools. These apps can enable healthcare providers to provide care to patients in remote or underserved areas, and can improve patient outcomes through early detection and intervention [52],[53],[54],[55].

Electronic health records (EHRs) and personal health records (PHRs)

These apps point to allow patients and healthcare providers to access and manage health information, such as medical history, test results, and medication records. EHRs and PHRs can improve the coordination of care between healthcare providers, and enable patients to take a more active role in their own healthcare [56],[57],[58].

Medical education and training apps

Supporting medical education and training, such as providing access to medical textbooks and resources, facilitating remote medical education, and supporting continuing medical education for healthcare professionals [59],[60].

Health Information Systems

HIS support the management and analysis of health data, such as population health management, public health surveillance, and health system performance monitoring [61],[62].

2.3.2 Values & Costs in Healthcare

Mobile medical apps can provide a range of values and benefits in healthcare, but they also come with costs and potential risks. Some of the values and costs associated with mobile medical apps in healthcare include the below listed categories.

VALUES OF MEDICAL MOBILE APPS

Improved patient outcomes

Mobile medical apps can support patient self-management and provide access to health information and resources, leading to better patient outcomes and quality of life.

There are several ways in which medical mobile apps can lead to improved patient outcomes:

1. **Increased patient engagement**: Medical mobile apps can help patients to take a more active role in managing their health. By providing patients with personalized information and tools for tracking their health, these apps can help patients to better understand their conditions and make more informed decisions about their care.
2. **Improved medication adherence**: Medical mobile apps can help patients to remember to take their medications on time, which can improve medication adherence and reduce the risk of complications.
3. **Early detection of health problems**: Some medical mobile apps allow patients to track their symptoms and vital signs, which can help to detect health problems early on. This can lead to earlier interventions and better outcomes.
4. **Improved communication with healthcare providers**: Many medical mobile apps allow patients to communicate with their healthcare providers, which can improve care coordination and reduce the risk of medical errors.
5. **Increased access to healthcare**: Medical mobile apps can provide patients with access to healthcare resources and services that they may not have otherwise been able to access. This can lead to better health outcomes for patients who may have difficulty accessing traditional healthcare services.

Cost savings

Mobile medical apps can reduce healthcare costs by enabling remote monitoring and telemedicine services, reducing hospital readmissions, and improving care coordination.

Here are a few ways in which medical mobile apps can help to reduce healthcare costs:

1. **Reduced hospital readmissions**: Patients can manage their conditions more effectively, which can reduce the need for hospital readmissions. This can lead to cost savings for healthcare providers and payers.
2. **Improved medication adherence**: As mentioned earlier, medical mobile apps can help to improve medication adherence. This can reduce the risk of complications and the need for costly medical interventions.
3. **Early detection of health problems**: By facilitating early detection of health problems, medical mobile apps can help to prevent more serious health issues from developing. This can reduce the need for expensive medical treatments and hospitalizations.

4. **Remote patient monitoring**: Some medical mobile apps allow healthcare providers to monitor patients remotely. This can reduce the need for in-person visits and allow healthcare providers to intervene early if any issues arise. This can lead to cost savings for healthcare providers and payers.
5. **Improved care coordination**: Improving care coordination between healthcare providers and patients reducing the risk of medical errors and ensure that patients receive the most appropriate and cost-effective care.

Convenience and accessibility

Regarding the convenience and accessibility, mHealth apps can improve access to healthcare services and resources, especially for patients in remote or underserved areas.

Convenience and accessibility can be achieved by:

1. **24/7 access to health information**: Access to health information anytime and anywhere, making it easier for patients to stay informed about their health and manage their conditions.
2. **Virtual consultations**: Some medical mobile apps allow patients to have virtual consultations with healthcare providers, eliminating the need for in-person visits and saving patients time and travel expenses.
3. **Prescription renewals**: Renew their prescriptions electronically, saving them time and eliminating the need for a visit to the pharmacy.
4. **Health tracking**: Tracking patient symptoms, medications, and vital signs, providing patients with a convenient way to monitor their health.
5. **Reminder notifications**: Sending reminder notifications to patients for medications, appointments, and other healthcare-related activities, helping patients stay on track with their health management.

Patient engagement

Patient engagement can be enriched by enabling patients to take a more active role in their own healthcare, leading to better health outcomes and patient satisfaction.

Some ways in which medical mobile apps can promote patient engagement:

1. **Personalized information and resources**: Providing evidences to patients with personalized information and resources based on their health condition, preferences, and needs. This can empower patients to take a more active role in their own healthcare.
2. **Health tracking and monitoring**: End – To – End tracking of symptoms, medications, and vital signs. This can help patients to better understand their health and make more informed decisions about their care.
3. **Goal setting and progress tracking**: Achieving health goals and track progress can motivate patients to make positive changes to their health and stay engaged in their healthcare.
4. **Social support and community building**: Social support features and community building tools can connect patients together having similar health conditions and share experiences, tips, and resources. This can create a sense of community and provide patients with emotional support and motivation.
5. **Patient education and empowerment**: Educational resources and tools can help patients understand their health conditions and treatment options. This can empower them to make informed decisions about their healthcare.

COSTS OF MEDICAL MOBILE APPS

Involving a range of costs, depending on various factors such as development complexity, maintenance requirements, hosting needs, regulatory compliance, and marketing efforts. Development costs can include designing, coding, testing, and debugging the app, which can range from a few thousand dollars to hundreds of thousands of dollars.

Maintenance costs may include ongoing updates and bug fixes, while hosting costs may involve expenses associated with data storage and certain features. Regulatory compliance costs can add to the expenses if the app handles sensitive patient information, and marketing costs may be necessary to promote the app to attract users.

Security and privacy risks

Security and privacy risks, such as data breaches and unauthorized access to sensitive health information can pose potential security and privacy risks to consider when evaluating the cost of medical mobile apps:

1. **Data breaches**: mHealth apps can collect and store sensitive patient data, including personal health information. If this data is not properly secured, it could be vulnerable to hacking and data breaches.
2. **Data sharing**: Sharing patient data with third-party partners or advertisers, which could compromise patient privacy.
3. **Inadequate data encryption**: Encrypting patient data to protect it from unauthorized access. If the app does not use strong encryption methods, patient data could be vulnerable to theft or misuse.
4. **Malware and hacking**: Medical mobile apps can be susceptible and vulnerable to malware and hacking if they are not designed with strong security features. This could compromise patient data and potentially harm patients.
5. **User error**: Users can pose security and privacy risks if they do not use the app correctly. For example, if a patient uses a weak password or shares their login credentials with others, their data could be compromised.

Technical issues and limitations

Technical limitations, such as compatibility issues with different devices and operating systems, which can impact their effectiveness and usability.

There are several technical issues and limitations to consider when evaluating the cost of medical mobile apps:

1. **Device compatibility**: Patients should ensure that they have access to compatible devices before downloading or purchasing the app.
2. **Network connectivity**: The importance of a data plan or internet connection to use is required, should help patients evaluate the cost of data plans and ensure that they can access the app without incurring significant expenses.
3. **Technical glitches and bugs**: Technical glitches and bugs that can impact their functionality and usability and anyone can be aware of potential issues and ensure that the app is regularly updated and maintained by the app developer.
4. **Limited features and functionality**: Limited features and functionality, which can impact usefulness for patients evaluating the features and functionality of the app to ensure that it meets their needs and expectations.
5. **Integration with other healthcare systems**: Some medical mobile apps may not be fully integrated with other healthcare systems, which can impact their effectiveness and usefulness for patients. This can ensure that the app is compatible with their healthcare

providers and other healthcare systems to ensure seamless integration and care coordination.

Limited evidence of effectiveness

Many mobile medical apps lack rigorous scientific evidence to support their effectiveness and safety, which can raise concerns about their reliability and potential harm.

Here are a few factors to consider when evaluating the limited evidence of effectiveness associated with medical mobile apps:

1. **Lack of rigorous research**: mHealth apps have not been rigorously tested through clinical trials or other research methods, which can make it difficult to assess their effectiveness and impact on patient outcomes.
2. **Limited long-term data**: Giving more details about limited data on long-term outcomes, can conclude to the basic difficult to assess their impact on patient health and wellbeing over time.
3. **Limited data on specific patient populations**: Limited data on specific patient populations can impact their effectiveness and usefulness for those patients.
4. **Variation in app quality**: Medical mobile apps can vary widely in terms of quality, functionality, and usability. This can impact their effectiveness and usefulness for patients.
5. **App developer bias**: App developers may have a vested interest in promoting their app and may be biased in their assessment of its effectiveness and impact on patient outcomes.

Overall, the values and costs associated with mobile medical apps in healthcare will depend on a range of factors, including the specific app and its intended use, the user population, and the regulatory and security requirements. It's important for healthcare organizations and providers to carefully evaluate mobile medical apps before adopting them, and to ensure that they are secure, effective, and compliant with regulatory requirements.

2.3.3 New Technology

There are a variety of new technology medical mobile apps that have been developed in recent years, ranging from apps that help patients manage chronic conditions to those that provide telemedicine services and remote monitoring.

Artificial Pancreas

A closed-loop system that monitors blood glucose levels and automatically delivers insulin to patients with type 1 diabetes, which involve a fully automated process for adjusting insulin delivery based on the user's glucose levels. One example of this is the Control-IQ system from Tandem Diabetes Care, which uses a mobile app to connect the CGM device and insulin pump and adjust insulin delivery in real-time based on the user's glucose levels and other factors such as exercise and meals [63].

Wearable health monitors

Devices that are worn on the body to track vital signs such as heart rate, blood pressure, and sleep patterns. Real-Time Monitoring and Alerts of Mobile apps for wearable health monitors are also incorporating real-time monitoring and alerts to help users stay on top of their health. For example, some devices can alert users if their heart rate or blood pressure is outside of a normal range, or if they need to take medication at a specific time [64].

3D printing

Used to create customized implants and prosthetics, as well as surgical models for pre-operative planning. 3D printing is also being used in the medical field, and mobile apps are being developed to facilitate the design and printing of medical devices and implants. For example, a recent study published in the Journal of 3D Printing in Medicine demonstrated the feasibility of using a mobile app to design and print custom implants for craniofacial reconstruction [65].

Telehealth Platforms

These platforms enable remote consultations and virtual visits between patients and healthcare providers. One of the most significant advancements in telehealth platform technology has been the development of virtual visit capabilities. Patients can now connect with healthcare providers via video chat, eliminating the need for in-person visits for routine check-ups or follow-up appointments [66].

Minimally invasive surgical tools – Robotic surgery

Advanced surgical tools that allow surgeons to perform procedures with smaller incisions, reducing patient pain and recovery time. Robotics technology is being integrated into minimally invasive surgical tools, allowing for more precise movements and reducing the risk of human error. Mobile apps can be used to control these robotic surgical tools remotely, allowing for more flexibility and reducing the need for complex and expensive surgical equipment [67].

Wireless health sensors

Small, wireless sensors that can be implanted in the body to monitor vital signs or track medication adherence. Wireless health sensors can be integrated with wearable devices such as fitness trackers or smartwatches, allowing for continuous monitoring of health parameters such as heart rate, blood pressure, and glucose levels [68].

Augmented reality (AR) and virtual reality (VR)

Used in medical training and education, as well as to help patients visualize their condition and potential treatments. Augmented Reality (AR) and Virtual Reality (VR) Integration: AR and VR technologies are being integrated into mobile apps for 3D printing to enhance the design and printing process. For example, some apps allow users to view 3D models in AR, which provides a more immersive experience and allows for better visualization of the finished product [69].

2.4 Related Work

Security analysis of mobile applications has been an active research area in recent years, due to the widespread adoption and usage of mobile devices and apps. Several studies have proposed various methods and tools for detecting and mitigating security vulnerabilities in mobile apps.

Janaka Senanayake et al. proposed a method to develop secure Android applications by mitigating code vulnerabilities with machine learning. They demonstrated that machine learning can assist in identifying and mitigating code vulnerabilities in Android apps, thereby improving the security of the applications [70]. Ashwag Albakri et al. conducted a survey on reverse-engineering tools for Android mobile devices. They discussed various reverse-engineering tools used for Android app analysis and presented a comparative analysis of the most popular tools [71]. Hilmi Abdullah and Subhi RM Zeebaree conducted a comprehensive review of Android mobile application vulnerabilities and prevention methods. They discussed the different types of vulnerabilities that exist in Android apps and presented various prevention methods to mitigate these vulnerabilities

[72]. Juliza Mohamad Arif et al. proposed a fuzzy analytic hierarchy process-based method to detect Android mobile malware. They demonstrated that their proposed method can effectively detect malware with high accuracy [73]. Gioacchino Tangari et al. analyzed the security issues of Android mobile health and medical applications. They identified the major security threats and vulnerabilities that exist in these applications and presented recommendations for improving their security [74].

Regarding to our analysis, the whole perspective covers the below three factors:

i) a malicious third-party app, which could be installed on a user's device alongside the vulnerable app and use its permissions to gain unauthorized access to sensitive data, ii) a physical attacker, who gains access to the user's device, either by stealing it or by accessing it while it is unattended, and iii) a remote attacker, who has intercepted the network traffic between the app and the relevant API.

Having in knowledge all the relevant security fields of analysis, some already published references pointed in the analysis of specific mHealth applications around some categories: manual inspection, dynamic analysis, static analysis, traffic analysis, insecure data storage, insecure network and inter-app communication [75],[76],[77].

After analyzed the existing work, none of them have conducted a comprehensive manual security analysis of multiple categories of medical apps on both Android and iOS platforms. The specific work also expands upon the threat model used in related research. Specifically, looked at the threat actor of a malicious third-party mobile app installed on the phone by developing a methodology that looks for side channel leaks that could allow third-party apps on the device to access sensitive data that belongs to the medical app. Finally, a good sample of 140 mHealth apps for manual analysis allowed us to provide a more accurate view of the security posture of such apps in both platforms of Android and iOS.

3 Methodology

In this chapter, is described the methodology used to analyze the security of Android applications using a set of open-source security tools. The tools used in this research include Mobile Security Framework (MobSF), Frida, Java Debugger (JDB), Android Debug Bridge (ADB), Apktool, BurpSuite and Jarsigner. These tools were used in combination to obtain a comprehensive understanding of the security of Android applications.

The objective of this research is to evaluate the effectiveness of these tools in identifying and mitigating security vulnerabilities in Android applications. A set of Android applications have been selected from the Google Play Store using a random sampling technique. These applications cover a diverse range of categories, including games, social networking, finance, and productivity. Virtual environment is needed to be implemented that replicates the Android operating system to carry out the security analysis.

3.1 Security Tools

In this thesis, some security tools have been used for security analysis. In the below chapters some key information is categorized per security tool and steps performed.

3.1.1 Android Debug Bridge

The Android Debug Bridge (ADB) [78], is a versatile command-line tool that allows developers and advanced users to interact with an Android device or emulator from a computer. ADB is included in the Android SDK (Software Development Kit) and can be installed on Windows, macOS, and Linux operating systems.

ADB has several features that make it a powerful tool for debugging and testing Android applications. Here are some of its key features:

Android Emulator using 7.1 version is running as below (Figure 3), explaining also basic usages:

```
C:\Users\Chris\AppData\Local\Android\Sdk\emulator>emulator.exe -writable-system -avd Android_7.1.1_API_25_
INFO    | Android emulator version 31.3.13.0 (build_id 9189900) (CL:N/A)
emulator: INFO: Found systemPath C:\Users\Chris\AppData\Local\Android\Sdk\system-images\android-25\google_apis\x86\
emulator: INFO: Found systemPath C:\Users\Chris\AppData\Local\Android\Sdk\system-images\android-25\google_apis\x86\
INFO    | Duplicate loglines will be removed, if you wish to see each indiviudal line launch with the -log-nofilter flag
.
```

Figure 3: Android Emulator 7.1 version is running with writable permissions.

1. **File transfer**: ADB can transfer files between the Android device and the computer. This can be useful for installing apps, copying logs, and retrieving data.
2. **Debugging**: ADB can connect to an Android device or emulator and provide debugging capabilities for developers. This allows developers to examine the device's log files, view real-time system stats, and diagnose problems with their apps.
3. **Shell access**: ADB can open a shell on the device or emulator, giving developers and advanced users direct access to the device's command-line interface. In addition, it's use can be expanded to various tasks, such as starting and stopping services, modifying system files, and installing apps (Figure 4).

```
C:\Users\Chris\AppData\Local\Android\Sdk\platform-tools>adb shell
generic_x86:/ # ls
acct          dev                  init.goldfish.rc         mnt              selinux_version     ueventd.rc
bugreports    etc                  init.ranchu-encrypt.rc   oem              sepolicy            var
cache         file_contexts.bin    init.ranchu-noencrypt.rc proc             service_contexts    vendor
charger       fstab.goldfish       init.ranchu.rc           property_contexts storage
config        fstab.ranchu-encrypt init.rc                  root             sys
d             fstab.ranchu-noencrypt init.usb.configfs.rc    sbin             system
data          init                 init.usb.rc              sdcard           ueventd.goldfish.rc
default.prop  init.environ.rc      init.zygote32.rc         seapp_contexts   ueventd.ranchu.rc
```

Figure 4: ADB shell and mobile details in directory.

4. **Screen capture**: ADB can capture screenshots of the device's screen and save them to the computer, explaining the ease for documenting app behavior or testing user interfaces.
5. **Port forwarding**: ADB can forward ports between the device and the computer, allowing developers to test network-based apps and services.

3.1.2 Frida

Frida is an open-source dynamic instrumentation toolkit that is widely used for security analysis of Android applications [79]. It allows security researchers and penetration testers to perform real-time manipulation of a running application's code and behavior.

Frida works by injecting a small JavaScript code into the target application at runtime. This code can then interact with the application's memory, hooks and alters function calls, and even replace entire functions with custom code. With Frida, you can perform a range of dynamic security analysis techniques on Android applications, including:

1. **Code injection**: Frida allows you to inject custom code into the running application, which can be used to perform various security analysis tasks. For example, you can use Frida to dump sensitive data from memory or modify the application's behavior to bypass security controls.
2. **Function tracing**: Frida can trace function calls made by the target application and provide real-time information about the arguments passed to each function. This can be useful for identifying vulnerabilities such as SQL injection or buffer overflow attacks.
3. **Hooking system calls**: Frida can hook into system calls made by the application and intercept data transmitted over the network. Explaining it’s effectiveness for analyzing network traffic and identifying potential security issues.
4. **Dynamic instrumentation**: Frida can dynamically instrument the application's code to identify vulnerabilities such as insecure file operations or unsecured communications.
5. **Reverse engineering**: Frida can be used in conjunction with other tools to reverse engineer the target application's code and identify vulnerabilities.

Frida installation in Android Emulator

As it can be listed below (Figure 5), the special Frida server file is pushed to emulator device in order to be executed.

```
C:\Users\Chris\AppData\Local\Android\Sdk\platform-tools>adb push frida-server-16.0.8-android-x86 /data/local/tmp
frida-server-16.0.8-android-x86: 1 file pushed, 0 skipped. 65.5 MB/s (53604060 bytes in 0.780s)
```

Figure 5: Push Frida server file in android emulator device.

Frida server give permissions and execute

Giving extra permissions needed before executed (Figure 6).

```
C:\Users\Chris\AppData\Local\Android\Sdk\platform-tools>adb shell
generic_x86:/ # cd data/local/tmp
generic_x86:/data/local/tmp # chmod 755 frida-server-16.0.8-android-x86
generic_x86:/data/local/tmp # ./frida-server-16.0.8-android-x86
```

Figure 6: Giving extra permissions needed for the execution of server file.

Frida hooking with specific methods of running applications

As first step, a monitoring process for running applications was performed (Figure 7):

```
┌──(kali㉿DESKTOP-4R2G87Q)-[/mnt/c/WINDOWS/system32]
└─$ frida-ps -aU
 PID  Name                          Identifier
----  ----------------------------  ------------------------------------
5588  Ada                           com.ada.app
5741  Chrome                        com.android.chrome
5265  Files                         com.android.documentsui
5233  Gallery                       com.android.gallery3d
4809  Gmail                         com.google.android.gm
2167  Google                        com.google.android.googlequicksearchbox
2167  Google                        com.google.android.googlequicksearchbox
5097  Hangouts                      com.google.android.talk
2198  Medical ID (free)             app.medicalid.free
1769  Microsoft SwiftKey Keyboard   com.touchtype.swiftkey
```

Figure 7: Frida monitoring process is running for active applications.

After that, and with the proof of app running, using Frida like below, method or function hooking should be performed (Figure 8):

Figure 8: Frida hooking of specific method running found.

3.1.3 Mobile Security Framework

In this thesis for the aspects of security analysis, MobSF framework was used for the purposes of statis analysis in both android and iOS application files [80].

Mobile Security Framework meaning

MobSF (Mobile Security Framework) is an open-source, automated mobile application security testing tool. It is designed to simplify the process of mobile application security testing, and it supports both Android and iOS platforms.

MobSF provides a wide range of features to identify and assess potential security vulnerabilities in mobile applications. Some of the key features of MobSF include dynamic and static analysis, binary analysis, malware analysis, and vulnerability assessments.

In terms of dynamic analysis, MobSF can intercept network traffic between the mobile app and the server to help identify potential vulnerabilities. It also includes a range of static analysis features, such as identifying insecure storage, detecting hard-coded secrets, and finding vulnerable code.

Another key feature of MobSF is its ability to conduct binary analysis of mobile applications. This includes analyzing the code of the application to identify potential vulnerabilities and to verify that the app is free from malware.

Finally, MobSF provides a range of vulnerability assessment features that can help identify potential security issues in an application. This includes identifying common vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure data storage.

Overall, MobSF is a powerful and comprehensive mobile application security testing tool that can help developers and security professionals identify and remediate potential vulnerabilities in mobile applications.

Installation and prerequisites

MobSF was installed in virtual box kali Linux machine. Python and Java are needed for installation and the whole MobSF source code can be found from [80],[81].

Run and upload file for analysis

After the installation was completed, running the run.sh file as below (Figure 9):

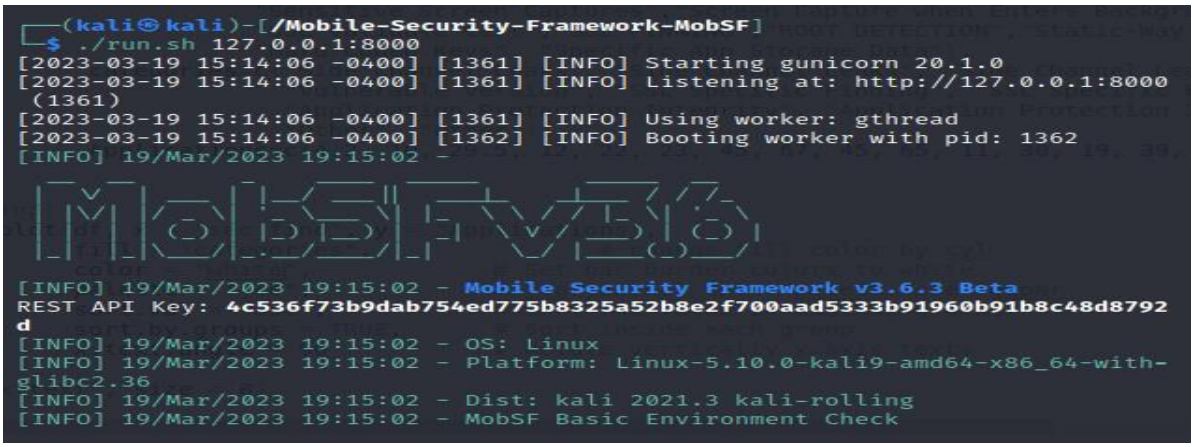


Figure 9: ModSF running on specific ip address and port.

After that, uploading of both APK and IPA files are required for the analysis.

Figure 10: MobSF uploading files screen.

MobSF Analysis categories Android

MobSF analysis categories for Android are described with fully details in the (Appendix 6.1)

MobSF Analysis categories iOS

MobSF analysis categories for iOS are described with fully details in the (Appendix 6.2)

3.1.4 Apktool

Apktool is a tool used for reverse engineering Android applications. It allows you to decode an APK file into its constituent resources and then recompile them back into a functional APK. This can be useful for analyzing the inner workings of an app, identifying security vulnerabilities, and even modifying the app's behavior [82].

Here are some of the features of Apktool:

1. **Decompiling APK files**: Apktool can decode an APK file and extract its resources, including the manifest file, XML files, and various binary files.

2. **Recompiling APK files**: Once you have modified an app's resources, you can use Apktool to recompile them into a new APK file.
3. **Smali/Baksmali support**: Apktool uses Smali and Baksmali, which are assembly languages for the Android Dalvik Virtual Machine, to disassemble and reassemble an app's bytecode.
4. **Debugging**: Apktool can generate debuggable APK files, which can be helpful for debugging an app or analyzing its behavior.
5. **Resource injection**: Apktool allows you to inject resources into an app, which can be useful for modifying an app's behavior or adding new functionality.

3.1.5 JarSigner

Jarsigner is a command-line tool provided by the Java Development Kit (JDK) that is used to digitally sign Java Archive (JAR) files, including Android Package (APK) files. Digital signatures are used to verify the authenticity and integrity of software, and are an important security feature of Android apps [83].

To use jarsigner for an APK file, you must first obtain a code signing certificate, which can be obtained from a certificate authority or created using the keytool utility that comes with the JDK.

Signing your APK file with jarsigner is an important step in the app development process, as it helps to ensure that your app is secure and can be trusted by users. It is particularly important when publishing your app on the Google Play Store, as Google requires all APK files uploaded to the store to be signed with a digital signature.

3.1.6 BurpSuite

Burp Suite is a software tool used for testing the security of web applications. It is developed by PortSwigger, a UK-based software company. Burp Suite helps in identifying vulnerabilities in web applications by intercepting and analyzing the traffic between the web application and the client [84].

Installation of Burp Suite on Android and iOS devices is different than installing it on a desktop or laptop computer. Burp Suite works as a proxy server that intercepts and analyzes the network traffic between a web application and the client. This functionality is available on both Android and iOS devices. When you start Burp Suite and configure your device's Wi-Fi connection to use the Burp Suite proxy server, all network traffic is routed through the Burp Suite proxy. This allows Burp Suite to monitor and analyze all incoming and outgoing requests made by the web application. Burp Suite provides a user interface that allows you to view and analyze the captured network traffic. You can view the HTTP requests and responses, including the headers and message body, and analyze them for any security vulnerabilities or issues.

For iOS devices, Burp Suite also provides the ability to intercept and analyze HTTPS traffic. This is achieved by installing the Burp Suite CA Certificate on the device, which allows Burp Suite to act as a trusted man-in-the-middle (MITM) and intercept encrypted traffic.

3.2 Analysis Categories

3.2.1 Side Channel Leaks

Side Channel Leaks are categorized in five discrete categories:

- **Third-party Keyboards:** One-way third-party keyboards can be vulnerable is through keylogging, which is the process of recording every keystroke made on a device. Some

third-party keyboards have been found to contain keylogging software, which can be used to capture sensitive information such as passwords and credit card numbers.

In addition, some third-party keyboards may request access to sensitive data or permissions that they do not need in order to function properly, which can also put users' data at risk. For example, a keyboard that requests access to a user's contacts or location data could potentially use that information for malicious purposes.

- **Copy-paste for sensitive fields:** One of the risks is the potential for malicious apps to read the clipboard data, which means that they could potentially access any sensitive information that was copied to the clipboard. This could happen even if the user has closed the app from which the information was copied.

  Another risk is the potential for apps to capture the screen while sensitive information is being displayed, which could allow attackers to view the information.
- **Sensitive Screen Captures**: If an app has access to the screen capture feature, it can potentially capture sensitive information, such as login credentials or credit card numbers, while the user is using the app.

  This can happen even if the user is not actively copying and pasting sensitive information, as the app may be able to capture the entire screen, including any sensitive data that is being displayed.
- **Captures**: When an app enters the background, the operating system may take a screenshot of the app's current state, which can include any sensitive information that is being displayed. So screenshot can then be accessible to other apps, the operating system itself, or even third-party services or apps that the user has granted permissions to.
- **Accessibility**: TalkBack is a feature designed to assist users with disabilities by providing spoken feedback as they interact with the user interface. However, if TalkBack is enabled in sensitive fields such as password fields, it can read aloud sensitive information, such as passwords or credit card numbers, as the user is typing them in.

  This can potentially expose the sensitive information to anyone who is within earshot of the device, or if the device is being recorded.

3.2.2 SSL Specific Findings

In this chapter an analysis of the SSL categories that checked in our analysis is performed.

- **Plaintext HTTP Requests – No SSL:** Plaintext HTTP requests in mobile apps are vulnerable because they are not encrypted and can be intercepted and read by third parties. Attackers are eligible to access sensitive information such as login credentials, personal information, and other data that should be kept private. Therefore, it is recommended that mobile apps use HTTPS (HTTP Secure) for all requests to encrypt the data being transmitted and ensure the security and privacy of their users' data.
- **SSL Pinning:** SSL pinning is a technique used in mobile apps to enhance the security of HTTPS connections. It involves hard-coding or "pinning" the SSL certificate of a specific server into the app, rather than relying on the default trust model of the device's operating system.

  By pinning the SSL certificate, the app will only trust that specific certificate and reject all others, even if they are signed by a trusted certificate authority. This helps to prevent man-in-the-middle attacks, where an attacker intercepts and modifies HTTPS traffic by using a fake SSL certificate. SSL pinning can be implemented in two ways: certificate pinning and public key pinning. Certificate pinning involves hard-coding the entire SSL certificate of the server into the app, while public key pinning only hard-codes the public key portion of the certificate.

  Implementing SSL pinning in mobile apps can help to mitigate security risks such

as network eavesdropping, SSL stripping attacks, and other forms of man-in-the-middle attacks. However, it is important to note that SSL pinning can also make it more difficult to update SSL certificates, as the app will need to be updated with the new certificate information.

3.2.3 Application Protection Integrity

It refers to the ability of an app to ensure that it has not been modified or tampered with by unauthorized parties.

There are several techniques that can be used to protect the integrity of a mobile app, such as code obfuscation, checksums, and code signing. Code obfuscation involves transforming the app's code to make it more difficult to reverse engineer and understand, while checksums and code signing are used to ensure that the app's code has not been modified or tampered with.

Application integrity protection is important because mobile apps can be vulnerable to a wide range of security threats, including malicious code injection, man-in-the-middle attacks, and other forms of tampering. By implementing application integrity protection, mobile app developers can help to mitigate these threats and ensure that their app is secure and trustworthy.

In addition to application integrity protection, other security measures should also be implemented, such as secure data storage, secure communication protocols, and user authentication mechanisms. By taking a comprehensive approach to mobile app security, developers can help to ensure that their app is as secure as possible and can be trusted by their users.

In this thesis, the below methodology was used to remove SSL Pinning mechanism from a mobile application starting from the decompiling of the file:

Decompile using Apktool

Below are shown the required steps needed for the process of decompiling. In first step the basic process of decompiling the .apk file is taking place (Figure 11).

Figure 11: Apktool import framework for specific application file.

The new decompiled folder of the basic .smali files has been created and will be needed for the next step in order to proceed with basic code changes (Figure 12).

```
┌──(kali㉿DESKTOP-4R2687Q)-[/mnt/c/Users/Chris/Desktop/DIPLWMATIKH]
└─$ apktool d /mnt/c/Users/Chris/Desktop/DIPLWMATIKH/FibriCheck_2.5.0_apkcombo.com.apk -o /mnt/c/Users/Chris/Desktop/newFibri3
I: Using Apktool 2.5.0 on FibriCheck_2.5.0_apkcombo.com.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/kali/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Baksmaling classes3.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Copying META-INF/services directory
```

Figure 12: Apktool decompiling of .apk file.

Perform code change to .smali files

In this example the result is to remove SSL Pinning check code from .smali code files (Figure 13,14).

```
public final void check(String hostname, List<? extends Certificate> peerCertificates) throws SSLPeerUnverifiedException {
    Intrinsics.checkNotNullParameter(hostname, str: "hostname");
    Intrinsics.checkNotNullParameter(peerCertificates, str: "peerCertificates");
    check$okhttp(hostname, new CertificatePinner$check$1(this, peerCertificates, hostname));
}

public final void check$okhttp(String hostname, Function0<? extends List<? extends X509Certificate>> cleanedPeerCertificatesFn) {
    Intrinsics.checkNotNullParameter(hostname, str: "hostname");
    Intrinsics.checkNotNullParameter(cleanedPeerCertificatesFn, str: "cleanedPeerCertificatesFn");
    List<Pin> findMatchingPins = findMatchingPins(hostname);
    if (findMatchingPins.isEmpty()) {
        return;
    }
    List<? extends X509Certificate> invoke = cleanedPeerCertificatesFn.invoke();
```

Figure 13: Found SSL Pinning mechanism in java source files (Part 1).

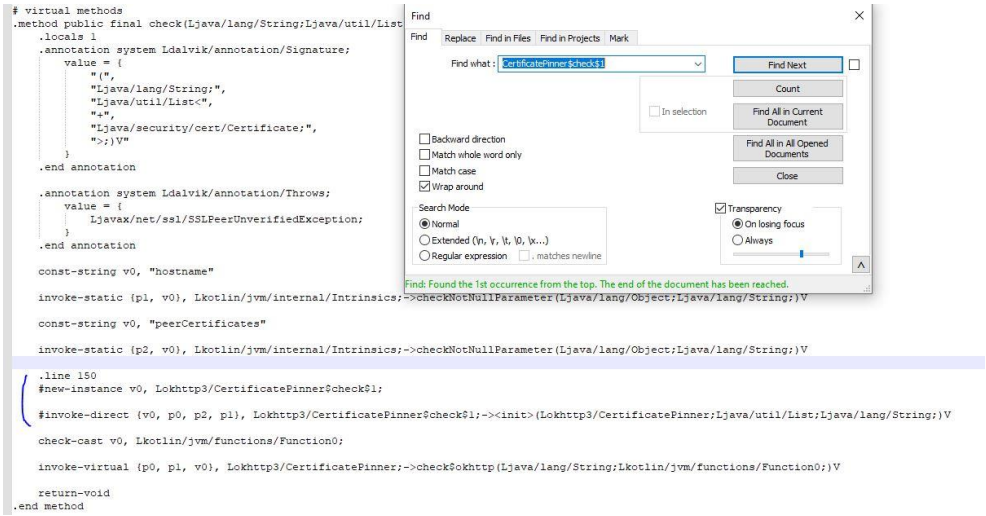


Figure 14: Found the exact point in code in smali files which will be removed for the new .apk file generation (Part 2).

Compile new .apk file with code change

New .apk file is created with the initiated code change of SSL Pinning removal (Figure 15).

```
┌──(kali㉿DESKTOP-4R2G87Q)-[/mnt/c/Users/Chris/Desktop]
└─$ apktool b newFibri3 -o fibriNew3.apk
I: Using Apktool 2.5.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes2 folder into classes2.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes3 folder into classes3.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Copying libs... (/kotlin)
I: Copying libs... (/META-INF/services)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

Figure 15: Apktool compile the new folder with code change and new .apk is generated.

Sign the new .apk file using JarSigner

The new .apk created should be signed using the custom sign files and jarSigner (Figure 16,17,18).

```
┌──(kali㉿DESKTOP-4R2G87Q)-[/mnt/c/Users/Chris/Desktop]
└─$ keytool -genkey -V -keystore /mnt/c/Users/Chris/Desktop/DIPLWMATIKH/key.keystore -alias chris -keyalg RSA -keysize 2
Enter keystore password:
```

Figure 16: Create using keytool new keystore file with RSA encryption.

```
┌──(kali㉿DESKTOP-4R2G87Q)-[/mnt/c/Users/Chris/Desktop]
└─$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore /mnt/c/Users/Chris/Desktop/DIPLWMATIKH/key.keystore fibriNew3.apk chris
Enter Passphrase for keystore:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/CHRIS.SF
  adding: META-INF/CHRIS.RSA
 signing: META-INF/services/kotlinx.coroutines.CoroutineExceptionHandler
 signing: META-INF/services/kotlinx.coroutines.internal.MainDispatcherFactory
 signing: AndroidManifest.xml
 signing: classes.dex
 signing: classes2.dex
 signing: classes3.dex
 signing: kotlin/annotation/annotation.kotlin_builtins
 signing: kotlin/collections/collections.kotlin_builtins
 signing: kotlin/coroutines/coroutines.kotlin_builtins
 signing: kotlin/internal/internal.kotlin_builtins
 signing: kotlin/kotlin.kotlin_builtins
 signing: kotlin/ranges/ranges.kotlin_builtins
 signing: kotlin/reflect/reflect.kotlin_builtins
 signing: lib/arm64-v8a/libbarhopper_v2.so
```

Figure 17: JarSigner basic command using sign file and apk file.

```
>>> Signer
    X.509, CN=chris mark, OU=unipi, O=unipi, L=Athens, ST=Greece, C=GR
    Signature algorithm: SHA256withRSA, 2048-bit key
    [trusted certificate]

jar signed.

Warning:
The signer's certificate is self-signed.
The SHA1 algorithm specified for the -digestalg option is considered a security risk and is disabled.
The SHA1withRSA algorithm specified for the -sigalg option is considered a security risk and is disabled.
```

Figure 18: JarSigner successfull result of operation.

Install new.apk file using adb to android device

After the new application is installed to the android emulator, then using BurpSuite, the login action is captured and then the user is successfully logged in using a new app which is custom signed and is deprived of SSL Pinning mechanism checking.

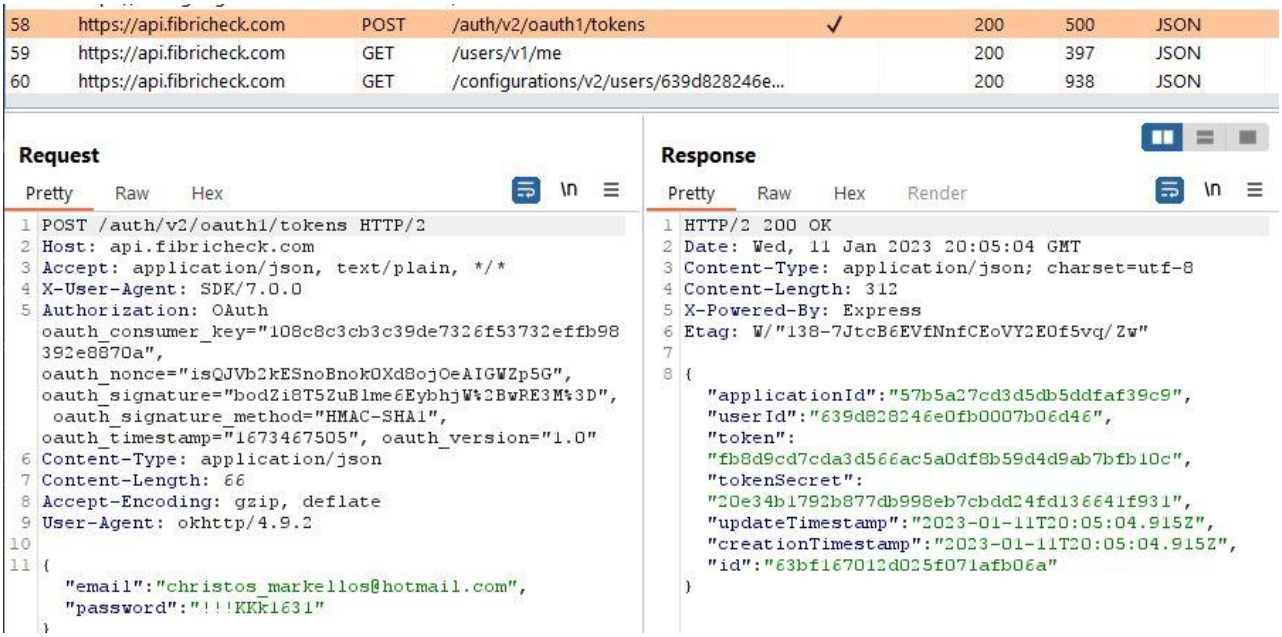


Figure 19: Result of successful removal of SSL Pinning mechanism and user successful logging using custom certificate.

3.2.4 Storing Data

Storing data on Android apps can be achieved through several mechanisms, depending on the type and size of the data, the security requirements, and the expected access patterns. Here are some of the common methods for storing data on Android apps:

1. **SharedPreferences**: This is a key-value pair storage mechanism that allows you to store small amounts of data, such as application preferences, settings, or user credentials. SharedPreferences are easy to use and do not require any complex setup. They are ideal for storing simple and non-sensitive data.

2. **SQLite Database**: SQLite is a lightweight and embedded database that provides a SQL interface for storing and querying data. It is well-suited for storing structured data, such as user profiles, contacts, or product catalog. SQLite can handle large amounts of data and supports transactions and indexing, making it suitable for complex data storage and retrieval.
3. **Content Providers**: Content Providers are a mechanism for sharing data between Android apps. They provide a standardized interface for storing and retrieving data that can be accessed by other apps, allowing for seamless integration and data sharing. Content Providers can be used to store and manage structured data, such as media files, contacts, or messages.
4. **File Storage**: Android apps can also store data as files on the device's file system. This can be useful for storing large files, such as multimedia or documents, that do not require frequent access or complex querying. File storage can be achieved using standard Java I/O libraries or Android-specific APIs, such as the External Storage or Internal Storage APIs.
5. **Cloud Storage**: For storing data that needs to be accessible across multiple devices or requires high availability and scalability, cloud storage services such as Google Drive, Dropbox, or Firebase can be used. These services provide easy-to-use APIs for storing and retrieving data, as well as robust security and backup mechanisms.

In this thesis in all .apks when the app is running on the device, additional check has been performed shared preferences files to identify if possible hardcoded keys or user credentials were stored without applying security perspectives (Figure 20).

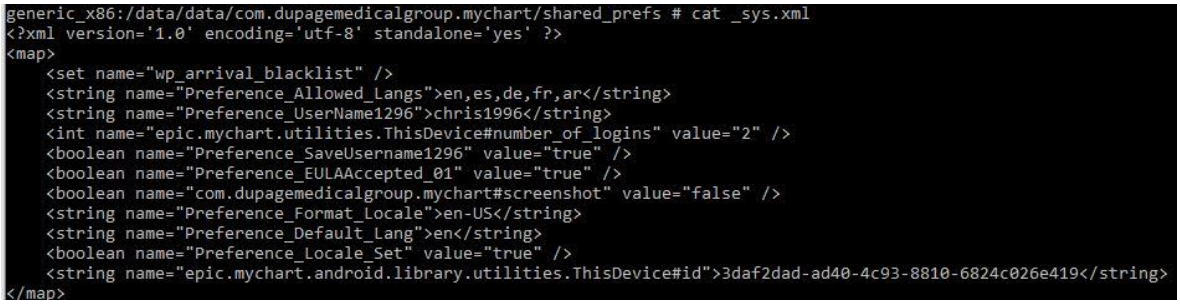


Figure 20: Exploring to /data/data in specific app shared_prefs folder and found user credentials.

3.2.5 Bypassing Root Detection

Bypassing root detection in Android and iOS apps refers to the process of circumventing or disabling the built-in security feature of an app that detects whether the user's device has been rooted (Android) or jailbroken (iOS).

Rooting and jailbreaking are the processes of removing restrictions imposed by the operating system on the device, which can enable users to access and modify system files and settings that are otherwise inaccessible. This can potentially lead to security risks, as well as give users unauthorized access to apps and services.

To prevent unauthorized access and protect the app from tampering, many app developers implement root detection as a security measure. When a rooted or jailbroken device is detected, the app may either restrict certain features or deny access altogether.

In the analysis that was performed only a few apps have applied restrictions to the users who may run apps in insecure rooted or jailbroken devices as below (Figure 21):

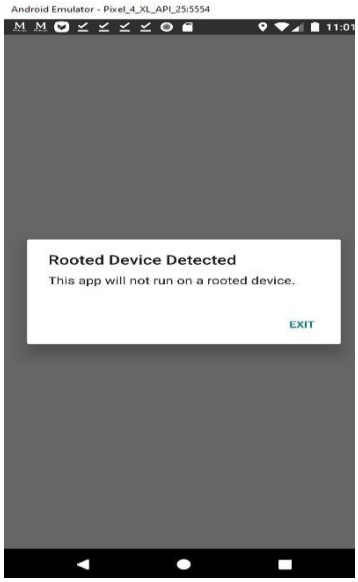


Figure 21: Root Detection mechanism found.

4 Results

After the part of the methodology and technical work has finished, resulting could take place in order to compare and contrast the crucial findings that found from analysis with the relevant theoretical aspects in the scientific literature and the desired result.

4.1 Thesis Categories for all analyzed applications – patient interaction

In this thesis was examined several applications with the key of belonging in different medical categories having also big downloading ratings from users.
The categories are proposed below, and are between patient and doctor interaction, pointing not only to medical library apps but also to both simulation and fitness apps.

4.1.1 Categories for analyzed applications

Mobile apps that were analyzed were 70 in total belonging also in 30 medical categories (Figure 22). The apps are in total 140 in number because were analyzed in both Android and iOS platforms.

App Category	# Examined
Allergy	2
Blood Donation	2
Cancer	3
Cardiovascular	3
Clinical	5
Dermatology	2
Diagnosis	1
Disease	1
Drug Information and Interactions	3
Endocrinology	2
Emergency Healthcare	1
Gastrointestinal	1
Geriatrics	1
Hacpatology	1
Health Insurance - Medical Library	7
Immunisation	1
Lifestyle Health and fitness	1
Maternity	1
Medication	4
Medicine Literature	2
Mental - Health	2
Neurology	2
Nutrition	2
Obstetrics and gynaecology	1
Orthopaedic	2
Other/Miscellaneous	10
Patient Examination	2
Radiology	1
Surgery	2
Telecommunication Doctors reservations	2
Total # of examined apps in both platforms (× 2)	140

Figure 22: Table of Apps Categories and number of apps examined.

4.1.2 Application with external device connection

There are some apps that were analyzed in which the external device connectivity is required for scanning or monitoring processes to take place to patients. In the below (Figure 23), can be identified that a percentage of 24.3% of totally analyzed applications has external patient interraction, pointing also to the ability of the user to to safe ensured of none data leakage can be found.

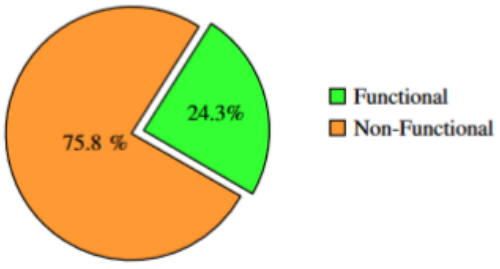


Figure 23: Mobile application with external device use support.

4.2 Thesis Application versions and ratings of downloads

In this chapter will be analyzed the belonging and the versions of the apps that was picked for the aspects of the analysis. Special version, vulnerable and not in both android and iOS platforms were picked and a download range up to 5000 was selected too.

4.2.1 Android / IOS versions of analyzed apps

The specific findings indicate that 10% of the Android apps support Android version 4, which is known to be vulnerable to various exploits. In addition, 61.43% support an Android version less than 5, 4.29% support Android 5.1, while 8.57% support Android 6, 14.29% support Android 7, 1.43% support Android 7.1, 5.7% apps support Android 8, and 4.29% apps support Android 9 (Figure 24).

The results is representative and raise the concern to the scientific and commercial community that application of big downloads rates use until now vulnerable Android 4 – 5 versions. Android 5.0 and earlier also lack important security features, such as full disk encryption and SELinux, which make them more susceptible to attacks. It is essential that app developers keep their apps up-to-date and support the latest secure versions of operating systems to ensure the privacy and security of their users.

Regarding iOS, we found that 11.43% of the medical apps still support iOS 9 or earlier versions (Figure 25). These older iOS versions lack important security features and are vulnerable to various exploits. Specifically, iOS 9 and earlier versions have known vulnerabilities that allow attackers to jailbreak the device, bypass passcode locks, and install malicious apps. Moreover, these older iOS versions do not receive security updates from Apple, making them more susceptible to attacks.

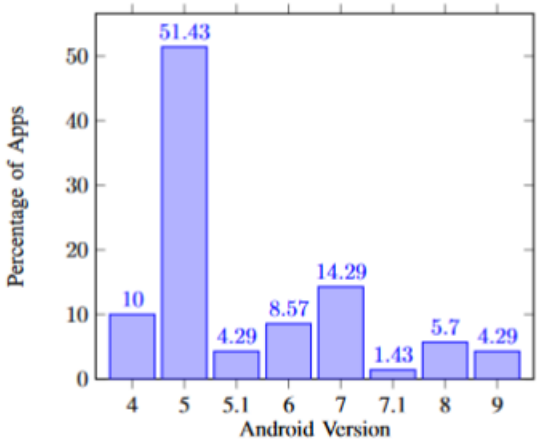


Figure 24: Android version vs percentage of apps used.

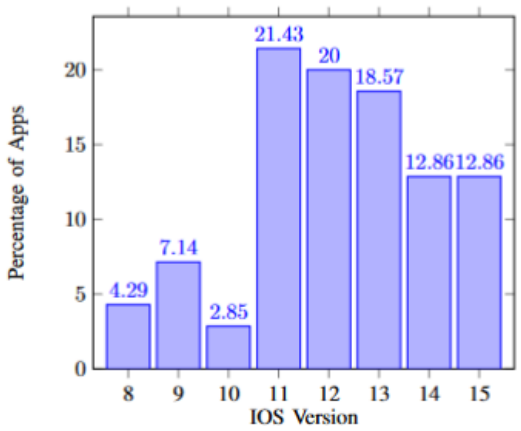


Figure 25: iOS versions vs percentage of apps used.

4.2.2 Number of downloads of medical applications

Regarding the popularity of each application installed and analyzed, below (Figure 26), propose a chart of Android applications pointing also that the most of them have been downloaded per customers having more than 100k downloads found in a percentage of 35,73% and among them 21.43% has more that 1M+ user downloads.

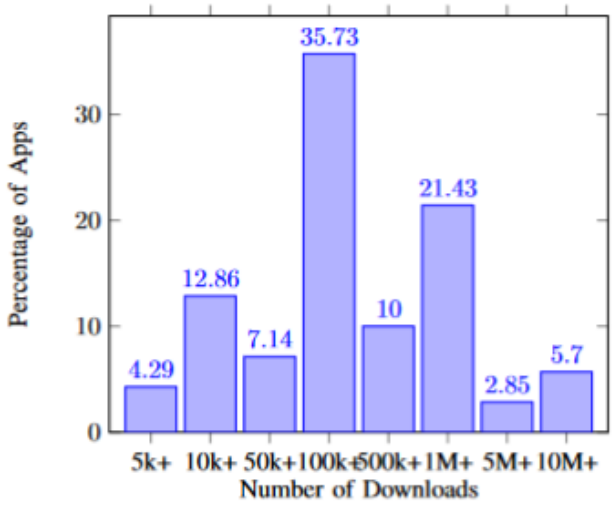


Figure 26: Android apps number of downloads vs percentage of the apps.

4.3 Metrics

In this chapter, the overall metrics about the proposed categories will be presented and evaluated regarding their aspect.

4.3.1 Side Channel Leaks

During the evaluation, identified that most of the evaluated apps lack safeguards against side - channel leaks, including pasteboard leakage, third-party keyboard leakage, and sensitive screen capture as illustrated in (Figure 27). These vulnerabilities could allow third-party applications installed on the device to circumvent sandbox restrictions and gain access to sensitive data, including PHI. Pasteboard leakage, for example, enables the user to copy sensitive information to the clipboard, which could be accessed by other applications installed on the device. Similarly, third-party keyboard leakage allows third-party applications to read keystrokes, including sensitive data such as passwords or private information, entered using the third-party keyboard. Sensitive screen capture leakage allows attackers to capture sensitive information displayed on the screen by taking screenshots or recording the screen. Additionally, Android and iOS automatically capture a snapshot when an application enters the background, which can be retrieved by other apps, further increasing the potential for data leakage.

Pointing further to the results, iOS has completely lower applications comparing the same ones of android, assuming that iOS applications are more secure regarding the initialization of third-party keyboards, screenshots and snapshots.

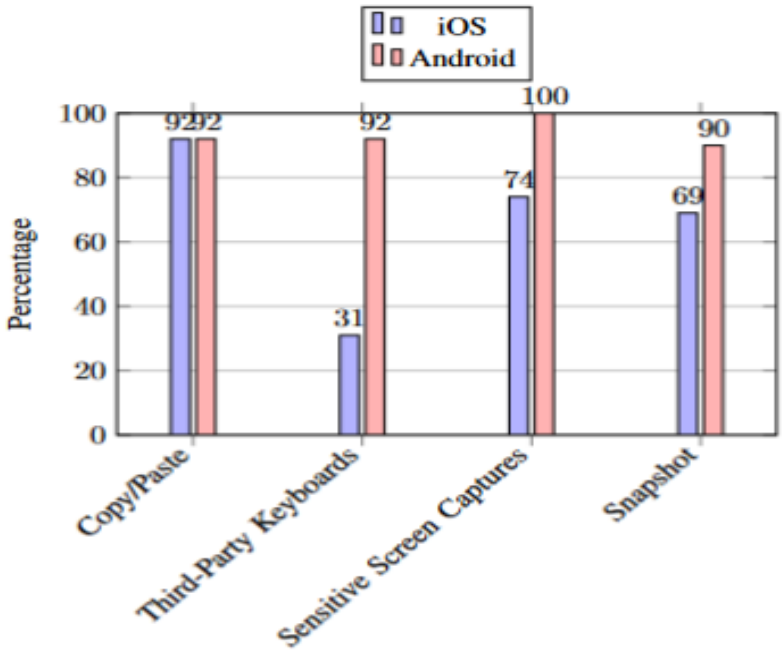


Figure 27: Side Channel Leaks percentage of Android and iOS apps.

4.3.2 SSL Pinning / Plain HTTP

The analysis was performed on whether the medical apps leveraged SSL or used plain HTTP, and whether SSL certificate validation was properly implemented. We also checked whether SSL Pinning
was implemented. We did find that two apps used plain HTTP (Figure 28), which is a critical issue given that these apps deal with PHI. Without the encryption provided by SSL, any data transmitted between the app and its server can be intercepted by an attacker who is able to eavesdrop on the communication. We also found that SSL Pinning was only implemented in 20% of the apps (Figure 28),

leaving the majority of the devices with a compromised trust store, vulnerable to man-in-the-middle attacks. SSL Pinning is an important security measure that ensures the authenticity of the SSL certificate presented by the server, even if the device's trust store has been compromised.

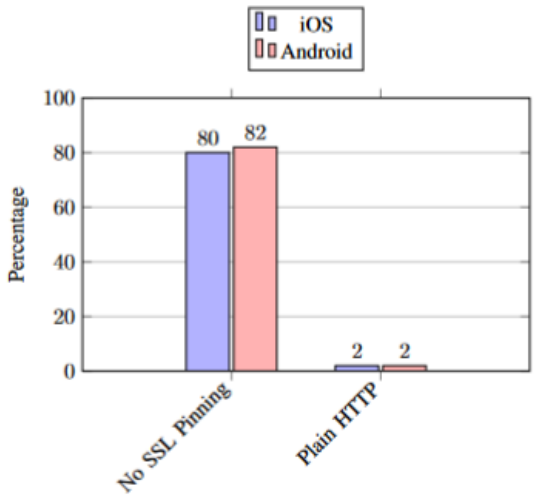


Figure 28: SSL findings of both Pinning and plain HTTP for Android and iOS apps.

4.3.3 Root – Jailbreak Detection

In order to assess the device integrity protection of the examined apps, was checked whether the apps implement root or jailbreak detection mechanisms. Out of the 70 Android apps, only 2 had root detection implemented, while only 3 out of 70 had jailbreak detection for iOS (Figure 29). The lack of device integrity protection exposes the apps to potential security threats, as a rooted or jailbroken device can compromise the security measures of the app and expose sensitive user data. Rooted or jailbroken devices can allow attackers to bypass security controls implemented in the app, such as SSL pinning or encryption mechanisms. Attackers can also use root access to install malware or manipulate the app's behavior. Without proper device integrity protection, sensitive information, such as Protected Health Information (PHI), can be exposed, leading to severe consequences for the user and the app's reputation.

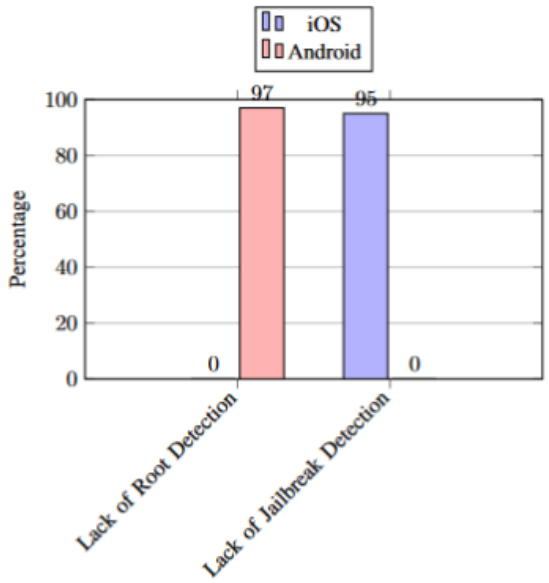


Figure 29: Lack of Jailbreak or Root detection for Android and iOS apps.

4.3.4 Application Integrity – obfuscation – ATS

Application Transport Security (ATS) - iOS

Out of the 70 iOS apps examined, found that 10 apps did not have Application Transport Security (ATS) enabled as illustrated in (Figure 30). ATS is a security feature in iOS that requires all app connections to use HTTPS with TLS 1.2 or higher, to ensure secure and encrypted communication. Without ATS, an attacker can potentially intercept network traffic and obtain sensitive user information. This is particularly concerning for medical apps that handle sensitive medical data, as an attacker can obtain this data by exploiting the lack of ATS.

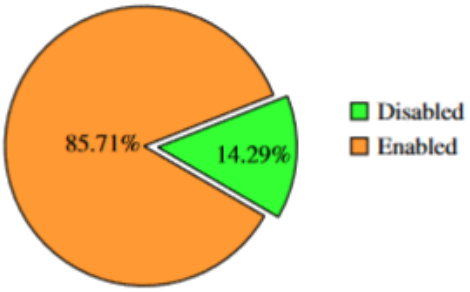


Figure 30: Application Transport Security per iOS apps in percentage.

Lack of Code Obfuscation – Android

Among the 70 Android apps examined in this study, 49 of them lacked any indication of code obfuscation as shown in (Figure 31). Code obfuscation is a critical security measure that safeguards an application's code from tampering and reverse engineering. The absence of proper code obfuscation presents a serious security risk, as attackers could easily decompile the app and

extract confidential information, including API and encryption keys. Moreover, it makes it easier for attackers to insert malicious code into the app, putting sensitive user data at risk.

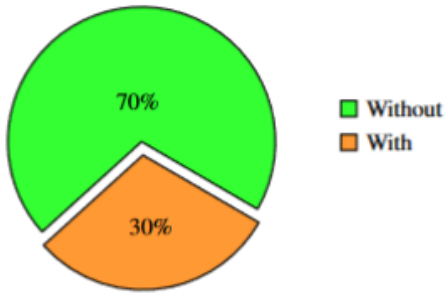


Figure 31: Lack of Obfuscation in androids apps in percentage.

4.3.5 Data Storage in Android Apps

Insecure local data storage was another major issue identified in our study. It is found that 17 out of the 70 medical apps in Android (24.28%) were insecurely storing sensitive data locally on the device, including usernames, passwords, session information, private IDs, PHI, and other medical data (Figure 32). The impact of this finding is significant, particularly in a rooted environment where third-party apps can access this data, bypassing the app's sandbox protections. This puts users' sensitive information at risk of theft or misuse, potentially leading to identity theft, financial fraud, or even physical harm in the case of medical data.

In the analysis perspective, it was noticed that some medical mobile applications had hardcoded sensitive secrets, such as API keys and passwords, which can be easily extracted by attackers. This finding is significant since hardcoded secrets can be exploited by attackers to gain unauthorized access to sensitive data or systems. Hardcoded secrets can also lead to the compromise of other systems and applications, which share the same secrets. Additionally, in a rooted device environment, attackers can extract these hardcoded secrets by bypassing sandbox protections, which can have severe consequences for the confidentiality and integrity of the data. Therefore, it is essential that developers avoid hardcoding sensitive secrets and implement secure storage mechanisms for these credentials.

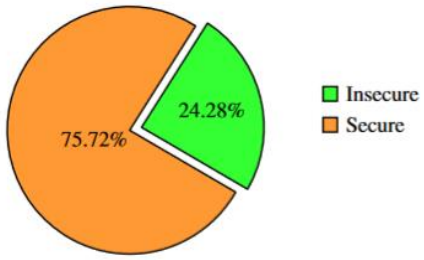


Figure 32: Data storage revealed secrets of android apps.

5 Conclusion

In conclusion, the analysis of 70 medical mobile applications across Android and iOS platforms has revealed that there are significant security risks associated with the use of these applications. While some apps demonstrated strong security measures, many others fell short in protecting sensitive user data.

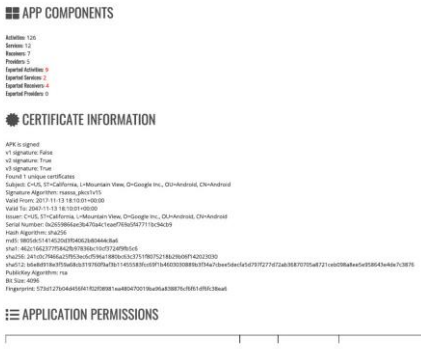
The findings of this study highlight the urgent need for mobile app developers, regulatory bodies, and healthcare providers to prioritize the security of medical mobile applications. Future studies in this area should focus on developing standardized security protocols for medical apps and implementing rigorous testing and certification processes to ensure that they meet these standards.

As the use of medical mobile applications continues to grow, it is essential that security measures keep pace with this trend. By addressing the security risks identified in this study and implementing robust security measures moving forward, we can ensure that medical mobile applications remain a valuable tool for improving healthcare outcomes while also protecting the privacy and security of users' sensitive information.

6 Appendices

6.1 MobSF Analysis Report Android Aetna App

Aetna Health (4.25.0.155763-prod)



PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
android.permission.CALL_PHONE	dangerous	directly call phone numbers	Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.
android.permission.INTERNET	normal	full internet access	Allows an application to create network sockets.
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission.USE_FINGERPRINT	normal	allow use of fingerprint	This constant was deprecated in API level 28. Applications should request USE_BIOMETRIC instead.
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected

			to and so on.
com.samsung.android.providers.context.permission.WRITE_USE_APP_FEATURE_SURVEY	unknown	Unknown permission	Unknown permission from android reference
org.fidoalliance.uaf.permissions.FIDO_CLIENT	unknown	Unknown permission	Unknown permission from android reference
android.permission.ACCESS_WIFI_STATE	normal	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.
android.permission.REORDER_TASKS	normal	reorder applications running	Allows an application to move tasks to the foreground and background. Malicious applications can force themselves to the front without your control.
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.
android.permission.READ_EXTERNAL_STORAGE	dangerous	read external storage contents	Allows an application to read from external storage.
android.permission.RECORD_AUDIO	dangerous	record audio	Allows application to access the audio record path.
android.permission.MODIFY_AUDIO_SETTINGS	normal	change your audio settings	Allows application to modify global audio settings, such as volume and routing.
android.permission.BLUETOOTH	normal	create Bluetooth connections	Allows applications to connect to paired bluetooth devices.

android.permission.WAKE_LOCK	normal	prevent phone from sleeping	Allows an application to prevent the phone from going to sleep.
android.permission.WRITE_CALENDAR	dangerous	add or modify calendar events and send emails to guests	Allows an application to add or change the events on your calendar, which may send emails to guests. Malicious applications can use this to erase or modify your calendar events or to send emails to guests.
android.permission.READ_CALENDAR	dangerous	read calendar events	Allows an application to read all of the calendar events stored on your phone. Malicious applications can use this to send your calendar events to other people.
android.permission.FOREGROUND_SERVICE	normal		Allows a regular application to use Service.startForeground.
android.permission.VIBRATE	normal	control vibrator	Allows the application to control the vibrator.
com.google.android.c2dm.permission.RECEIVE	signature	C2DM permissions	Permission for cloud to device messaging.
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE	unknown	Unknown permission	Unknown permission from android reference

APKID ANALYSIS

FILE	DETAILS	
	FINDINGS	DETAILS
classes.dex	Anti-VM Code	Build.FINGERPRINT check Build.MANUFACTURER check Build.TAGS check SIM operator check network operator name check
	Compiler	r8 without marker (suspicious)
classes2.dex	FINDINGS	DETAILS
	Anti-VM Code	Build.FINGERPRINT check Build.MODEL check Build.MANUFACTURER check Build.PRODUCT check Build.HARDWARE check Build.BOARD check possible Build.SERIAL check network operator name check device ID check
	Compiler	r8 without marker (suspicious)
classes3.dex	FINDINGS	DETAILS
	Anti-VM Code	Build.FINGERPRINT check Build.MANUFACTURER check Build.HARDWARE check Build.TAGS check device ID check
	Compiler	r8 without marker (suspicious)
classes4.dex	FINDINGS	DETAILS
	Compiler	r8 without marker (suspicious)
classes5.dex	FINDINGS	DETAILS
	Compiler	r8 without marker (suspicious)

BROWSABLE ACTIVITIES

ACTIVITY	INTENT
com.aetna.aetnahealth.activity.AetnaSplashScreenActivity	Schemes: aetnadigital://, Hosts: login_callback,
com.aetna.aetnahealth.activity.InnovationHealthSplashScreenActivity	Schemes: aetnadigital://, Hosts: login_callback,
com.aetna.aetnahealth.activity.BannerHealthSplashScreenActivity	Schemes: aetnadigital://, Hosts: login_callback,
com.aetna.aetnahealth.activity.SutterHealthSplashScreenActivity	Schemes: aetnadigital://, Hosts: login_callback,
com.aetna.aetnahealth.activity.TexasHealthSplashScreenActivity	Schemes: aetnadigital://, Hosts: login_callback,
com.aetna.aetnahealth.activity.AllinaHealthSplashScreenActivity	Schemes: aetnadigital://, Hosts: login_callback,
	Schemes: https://, aetnadigital://,
com.aetna.aetnahealth.activity.DeepLinkActivity	Hosts: health.aetna.com, open, aetna.app.link, aetna-alternate.app.link, Path Prefixes: /saladoc, /actions, /manage/claims, /manage/coverage-requests, /benefits/medical-plan-summary,

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION
1	www.aetna.com api1.aetna.com api2.aetna.com api3.aetna.com adtm-cdn.aetna.com content.prd.aetnadigital.com	secure	Certificate pinning does not have an expiry. Ensure that pins are updated before certificate expire. [Pin: 0rv4XQw5pZni/0CBFcQjhSjNEzvghB5GUVNKhM=UmQE= Digest: SHA-256,Pin: NIgvyyhcGFo7vIL/KJw6cn8iFWO9fJtWrIl/J7OSHMI= Digest: SHA-256]

CERTIFICATE ANALYSIS

TITLE	SEVERITY	DESCRIPTION
Signed Application	info	Application is signed with a code signing certificate

MANIFEST ANALYSIS

NO	ISSUE	SEVERITY	DESCRIPTION
1	App has a Network Security Configuration	info	The Network Security Configuration feature lets apps customize their network security settings in a safe, declarative configuration file without modifying app code.
	[android:networkSecurityConfig=@xml/network_security_config]		These settings can be configured for specific domains and for a specific app.
2	Broadcast Receiver (com.appboy.AppboyGcmReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: com.google.android.c2dm.permission.SEND [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
3	Broadcast Receiver (com.google.firebase.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: com.google.android.c2dm.permission.SEND [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
4	Broadcast Receiver (com.google.android.gms.measurement.AppMeasurementInstallReferrerReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.INSTALL_PACKAGES [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
			A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission
5	Broadcast Receiver (androidx.profileinstaller.ProfileInstallReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	warning	which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

CODE ANALYSIS

NO	ISSUE	SEVERITY	STANDARDS	FILES
				a/a.java com/accepttо/eguardianframework/utils/CallbackHolder.java com/activeandroid/ActiveAndroid.java com/activeandroid/Cache.java com/activeandroid/Configuration.java com/activeandroid/DatabaseHelper.java com/activeandroid/Model.java com/activeandroid/ModelInfo.java com/activeandroid/TableInfo.java com/activeandroid/query/From.java com/activeandroid/util/IOUtils.java com/activeandroid/util/Log.java com/activeandroid/util/ReflectionUtils.java com/activeandroid/util/SQLiteUtils.java com/adobe/marketing/mobile/AbstractHitsDatabase.java com/adobe/marketing/mobile/AnalyticsExtension.java com/adobe/marketing/mobile/AnalyticsHitsDatabase.java com/adobe/marketing/mobile/AnalyticsListenerAcquisitionResponseContent.java com/adobe/marketing/mobile/AnalyticsListenerAnalyticsRequestContent.java

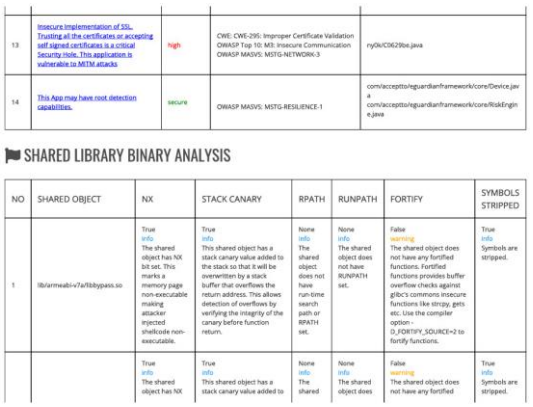
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	com/hypr/hyprandroidsuladapter/ui/screens/settingsbiometrics/HyprSettingsBiometricsPresenter.java com/hypr/hyprandroidsuladapter/ui/views/CircleOverlayView.java com/hypr/hyprandroidsuladapter/useragent/HyprUafUserAgent.java com/hypr/hyprandroidsuladapter/useragent/HyprUserAgentOperation.java com/konylabs/android/C0202a.java com/konylabs/android/C0205d.java com/konylabs/android/C0209h.java com/konylabs/android/C0212k.java com/konylabs/android/C0215n.java com/konylabs/android/G.java com/konylabs/android/KonyApplication.java com/konylabs/android/KonyBootCompletedReceiver.java com/konylabs/android/KonyMain.java com/konylabs/android/RunnableC0204c.java com/konylabs/android/RunnableC0210i.java com/konylabs/android/RunnableC0211j.java com/konylabs/android/r.java com/konylabs/api/A.java com/konylabs/api/B.java com/konylabs/api/C.java com/konylabs/api/C0222ae.java com/konylabs/api/C0237k.java com/konylabs/api/C0238l.java com/konylabs/api/C0239m.java com/konylabs/api/C0240n.java com/konylabs/api/C0241o.java com/konylabs/api/C0247u.java com/konylabs/api/C0518v.java com/konylabs/api/C0559w.java com/konylabs/api/C0562z.java com/konylabs/api/E.java com/konylabs/api/F.java com/konylabs/api/J.java com/konylabs/api/M.java com/konylabs/api/R.java com/konylabs/api/RunnableC0234h.java com/konylabs/api/S.java

2	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	warning	CWE: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality	com/adobe/marketing/mobile/LegacyAbstractHitDatabase.java com/konylabs/api/C0247u.java com/konylabs/api/db/b.java com/konylabs/api/db/e.java com/konylabs/api/db/f.java com/konylabs/js/api/H.java com/konylabs/js/api/O.java com/konylabs/notification/KonyLocalNotificationBroadcastReceiver.java com/konylabs/notification/a.java com/konylabs/notification/c.java com/newrelic/agent/android/instrumentation/SQLiteInstrumentation.java ny0k/C0612ao.java ny0k/C0613ap.java ny0k/C0640bp.java ny0k/aG.java
				com/activeandroid/Cache.java com/adobe/marketing/mobile/LegacyReferrerHandler.java com/aetna/aetnahealth/BuildConfig.java com/aetna/aetnahealth/auth/BsiIdentityInfo.java com/aetna/aetnahealth/data/remote/json/DesignatedQualityProgramsPayload.java com/aetna/aetnahealth/data/remote/json/InterstitialData.java com/aetna/aetnahealth/domain/models/FilterModel.java com/aetna/aetnahealth/manager/SharedPreferencesManager.java com/aetna/aetnahealth/manager/deeplinks/DeepLinkSchemaManifest.java com/aetna/aetnahealth/model/json/debug/PushPayload.java com/aetna/pharmacycomponents/json/CvsTokenResponse.java com/aetna/pharmacycomponents/json/SecurityContext.java com/aetna/pharmacycomponents/model/CvsToken

3	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	java com/aetna/retrofit/cache/ResponseCache.java com/bumptech/glide/load/engine/EngineResource.java com/hypr/hyprandroidcommon/iasm/HyprBaseAsmActivity.java com/hypr/hyprandroidcommon/useragent/data/dalastruct/BaseFragmentData.java com/hypr/hyprandroidcommon/useragent/data/fido/HyprActivityResultData.java com/hypr/hyprandroidcommon/useragent/data/fido/HyprOobData.java com/hypr/hyprandroidfidoframework/uafclient/activities/HyprBaseClientActivity.java com/hypr/hyprandroidfingerprintframework/ui/activities/HyprFingerprintActivity.java com/hypr/hyprandroidpinframework/ui/activities/HyprPinActivity.java com/konylabs/api/R.java com/konylabs/api/TableLib.java com/konylabs/api/ui/IK.java com/newrelic/agent/android/SavedState.java com/newrelic/agent/android/distributedtracing/TracePayload.java com/newrelic/agent/android/harvest/AgentHealth.java com/newrelic/agent/android/harvest/HarvestConfiguration.java com/newrelic/agent/android/measurement/producer/HttpErrorMeasurementProducer.java com/newrelic/agent/android/util/PersistentUUID.java com/opentok/android/DefaultAudioDevice.java com/teladoc/members/sdk/data/Action.java com/teladoc/members/sdk/data/Field.java com/teladoc/members/sdk/data/Screen.java com/teladoc/members/sdk/data/message/AttachmentData.java com/teladoc/members/sdk/data/message/ChatActionData.java com/teladoc/members/sdk/data/message/ChatMessageData.java

				com/teladoc/members/sdk/data/message/ChatParticipant.java com/teladoc/members/sdk/data/message/MessageRoomConfig.java com/teladoc/members/sdk/data/message/TooltipData.java com/teladoc/members/sdk/data/rows/TableRowData.java ny0k/C0612ao.java
4	This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel.	secure	OWASP MASVS: MSTG-NETWORK-4	com/aetna/aetnahealth/dagger/base/modules/RetrofitModule_BuildRetrofitFactory.java com/hypr/hyprandroidcommon/relyingpartyadapter/retrofit/RetrofitGenerator.java ny0k/C0629be.java ny0k/C0631bg.java ny0k/C0632bh.java ny0k/C0634bj.java ny0k/C0649by.java
5	The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks.	high	CWE: CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-3	com/acceptto/eguardianframework/activities/RBAAcceptToOutIDActivity.java com/teladoc/members/sdk/controllers/shared/PinScreenViewController.java
6	App can read/write to External Storage. Any App can read data written to External Storage.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	com/hypr/hyprandroidcommon/utils/HyprConformanceLogger.java com/konylabs/api/ui/C0432gw.java com/konylabs/api/ui/gG.java com/konylabs/js/api/C0578p.java com/newrelic/agent/android/AndroidAgentImpl.java com/teladoc/members/sdk/utils/ImageUploader.java com/teladoc/members/sdk/views/FormPhotoPicker.java ny0k/C0615ar.java ny0k/C0617at.java ny0k/fG.java

7	MD5 is a weak hash known to have hash collisions.	warning	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	com/adobe/marketing/mobile/LegacyRemoteDownload.java com/appboy/support/StringUtils.java com/pubnub/api/vendor/Crypto.java org/matomo/sdk/tools/Checksum.java
8	Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	warning	CWE: CWE-749: Exposed Dangerous Method or Function OWASP Top 10: M1: Improper Platform Usage OWASP MASVS: MSTG-PLATFORM-7	com/adobe/marketing/mobile/GriffonWebViewSocket.java com/appboy/ui/inappmessage/views/AppboyInAppMessageHtmlFullView.java com/teladoc/members/sdk/utils/WebViewFocusHelper.java com/teladoc/members/sdk/views/WebViewField.java
9	The App uses an Insecure Random Number Generator.	warning	CWE: CWE-330: Use of Insufficiently Random Values OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6	com/appboy/support/IntentUtils.java com/konylabs/api/C0242p.java com/konylabs/api/C0562z.java com/konylabs/nativecodegen/api/MathLib.java com/newrelic/agent/android/util/Util.java com/pubnub/api/vendor/Crypto.java org/matomo/sdk/Tracker.java
10	SHA-1 is a weak hash known to have hash collisions.	warning	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	com/hypr/hyprandroidcommon/authenticator/uaf/tlv/v1/BasicTlvUtils.java com/konylabs/api/C0238l.java com/newrelic/agent/android/harvest/HttpError.java com/pubnub/api/vendor/FileEncryptionUtil.java
11	App creates temp file. Sensitive information should never be written into a temp file.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	com/aetna/aetnahealth/presentation/securemessaging/SecureMessagingAttachmentManager$getCameraFileUri$2.java com/teladoc/members/sdk/utils/ImageUploader.java
12	App can write to App Directory. Sensitive Information should be encrypted.	info	CWE: CWE-276: Incorrect Default Permissions OWASP MASVS: MSTG-STORAGE-14	com/aetna/aetnahealth/manager/SharedPreferencesManager.java com/aetna/theming/ThemeManagerImpl.java com/konylabs/api/C0236j.java



13	Insecure implementation of SSL. Trusting all the certificates or accepting self signed certificates is a critical Security Hole. This application is vulnerable to MITM attacks	high	CWE: CWE-295: Improper Certificate Validation OWASP Top 10: M3: Insecure Communication OWASP MASVS: MSTG-NETWORK-3	ny0k/C0629be.java
14	This App may have root detection capabilities.	secure	OWASP MASVS: MSTG-RESILIENCE-1	com/acceptto/eguardianframework/core/Device.java com/acceptto/eguardianframework/core/RiskEngine.java

SHARED LIBRARY BINARY ANALYSIS

NO	SHARED OBJECT	NX	STACK CANARY	RPATH	RUNPATH	FORTIFY	SYMBOLS STRIPPED
1	lib/armeabi-v7a/libbypass.so	True info The shared object has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.	True info This shared object has a stack canary value added to the stack so that it will be overwritten by a stack buffer that overflows the return address. This allows detection of overflows by verifying the integrity of the canary before function return.	None info The shared object does not have run-time search path or RPATH set.	None info The shared object does not have RUNPATH set.	False warning The shared object does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets etc. Use the compiler option -D_FORTIFY_SOURCE=2 to fortify functions.	True info Symbols are stripped.
		True info The shared object has NX	True info This shared object has a stack canary value added to	None info The shared	None info The shared object does	False warning The shared object does not have any fortified	True info Symbols are stripped.

8	lib/arm64-v8a/libopentok.so	True info The shared object has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.	True info This shared object has a stack canary value added to the stack so that it will be overwritten by a stack buffer that overflows the return address. This allows detection of overflows by verifying the integrity of the canary before function return.	None info The shared object does not have run-time search path or RPATH set.	None info The shared object does not have RUNPATH set.	True info The shared object has the following fortified functions: ['__FD_SET_chk', '__memcpy_chk', '__memset_chk', '__FD_CLR_chk', '__FD_ISSET_chk', '__vsnprintf_chk', '__vsprintf_chk', '__read_chk', '__strchr_chk']	True info Symbols are stripped.

NIAP ANALYSIS v1.3

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION
1	FCS_RBG_EXT.1.1	Security Functional Requirements	Random Bit Generation Services	The application invoke platform-provided DRBG functionality for its cryptographic operations.
2	FCS_STO_EXT.1.1	Security Functional Requirements	Storage of Credentials	The application does not store any credentials to non-volatile memory.
3	FCS_CKM_EXT.1.1	Security Functional Requirements	Cryptographic Key Generation Services	The application implement asymmetric key generation.
4	FDP_DEC_EXT.1.1	Security Functional Requirements	Access to Platform Resources	The application has access to ['network connectivity', 'location', 'camera', 'bluetooth', 'microphone'].
5	FDP_DEC_EXT.1.2	Security Functional Requirements	Access to Platform Resources	The application has access to ['calendar'].
6	FDP_NET_EXT.1.1	Security Functional Requirements	Network Communications	The application has user/application initiated network communications.
24	FPT_TUD_EXT.2.1	Selection-Based Security Functional Requirements	Integrity for Installation and Update	The application shall be distributed using the format of the platform-supported package manager.
25	FCS_CKM.1.1(2)	Optional Security Functional Requirements	Cryptographic Symmetric Key Generation	The application shall generate symmetric cryptographic keys using a Random Bit Generator as specified in FCS_RBG_EXT.1 and specified cryptographic key sizes 128 bit or 256 bit.

DOMAIN MALWARE CHECK

DOMAIN	STATUS	GEOLOCATION
health.innovationhealthmedicare.com	ok	IP: 52.85.158.89 Country: Greece Region: Attiki City: Athens Latitude: 37.979450 Longitude: 23.716221 View: Google Map
www.health.aetna.com	ok	IP: 45.223.19.220 Country: France Region: Ile-de-France City: Paris Latitude: 48.853409 Longitude: 2.348800 View: Google Map
member.banneraetna.myplanportal.com	ok	IP: 206.213.253.39 Country: United States of America Region: Connecticut City: Middletown Latitude: 41.550903 Longitude: -72.663681
		View: Google Map
adcc-cdn.aetnadigital.com	ok	IP: 52.85.158.39 Country: Greece Region: Attiki City: Athens Latitude: 37.979450 Longitude: 23.716221 View: Google Map
config.opentok.com	ok	IP: 52.85.158.34 Country: Greece Region: Attiki City: Athens Latitude: 37.979450 Longitude: 23.716221 View: Google Map

FIREBASE DATABASES

FIREBASE URL	DETAILS
https://aetna-health.firebaseio.com	info App talks to a Firebase Database.

EMAILS

EMAIL	FILE
confidential@crypto.aetna	com/aetna/voltage/VoltageCryptoPayload.java
test@testemail.com	com/aetna/aetnahealth/presentation/nba/debug/NbaComponentDetailActivity.java
hipaaauthorization@aetna.com support@hypr.com	Android String Resource
appro@openssl.org	lib/arm64-v8a/libopentok.so

TRACKERS

TRACKER	CATEGORIES	URL
Adobe Experience Cloud		https://reports.exodus-privacy.eu.org/trackers/229
Branch	Analytics	https://reports.exodus-privacy.eu.org/trackers/167
Braze (formerly Appboy)	Analytics, Advertisement, Location	https://reports.exodus-privacy.eu.org/trackers/17
Google Firebase Analytics	Analytics	https://reports.exodus-privacy.eu.org/trackers/49
Matomo (Piwik)	Analytics	https://reports.exodus-privacy.eu.org/trackers/138
New Relic	Analytics	https://reports.exodus-privacy.eu.org/trackers/130

HARDCODED SECRETS

POSSIBLE SECRETS
"bad_token" : "Invalidating"
"com_appboy_api_key" : "c5b49cd6-4cba-4aef-9315-6410ffb1be63"
"contact_info_cell_key" : "Cell"
"contact_info_mobile_key" : "Mobile"
"expired_token" : "Expiring"
"fingerprint_tagging_screen_auth_failed" : "touch_id_authentication_failed_view"
"firebase_database_url" : "https://aetna-health.firebaseio.com"
"google_api_key" : "AIzaSyAeGZtMUdCbbHksfoVKJWD-0qUecrWZbnY"
"google_crash_reporting_api_key" : "AIzaSyAeGZtMUdCbbHksfoVKJWD-0qUecrWZbnY"
"hypr_tee_key_data" : "Data"
"hypr_tee_key_exists_false" : "False"
"hypr_tee_key_exists_true" : "True"
"hypr_tee_key_generate_assertion" : "Assertion:"
"hypr_tee_key_generate_challenge" : "Challenge:"
"hypr_tee_key_sign" : "Sign"
"hypr_tee_key_signature" : "Signature:"
"hypr_tee_key_signed_title" : "Signed"
"hypr_tee_key_type_rsa" : "RSA"
"password" : "Password"
"revoking_token" : "Revoking"
"use_fingerprint_to_authenticate_key" : "use_fingerprint_to_authenticate_key"

PLAYSTORE INFORMATION

Title: Aetna Health

Score: 4.688706 **Installs:** 1,000,000+ **Price:** 0 **Android Version Support:** **Category:** Medical **Play Store URL:** com.aetna.aetnahealth

Developer Details: Aetna Inc, Aetna+Inc, None, http://www.aetna.com, Aetna+Health-AppSupport@aetna.com,

Release Date: Dec 31, 2017 **Privacy Policy:** Privacy link

Description:

The Aetna Health℠ app helps you stay on top of your health care when and where it works for you. With just a few taps, you can find a doctor, see claims and plan information, and access your ID card. You can even talk with a doctor anytime by phone or video. It's simple. And, it's all in the palm of your hand. This app is available to most Aetna members. Members in select plans may not be eligible yet, but we're continually working to expand access. If you aren't eligible today, check back in a few months. App features may vary depending on your plan. Connect to care • Search for in-network providers and facilities • See provider ratings and reviews from other patients • Get cost estimates before you get care* • Look up medications by their brand or generic names and learn about side effects • Talk with a doctor anytime by phone or video* • Connect with COVID vaccine resources Manage your health and benefits • Access your ID card wherever you need it • Track spending and progress toward meeting your deductible* • Check PayFlex® HSA, HRA and FSA account balances* • View your plan summary and get information about what's covered* • See claims and Explanation of Benefits (EOBs) to get information about what's covered • Receive personalized health reminders* (Eligible members with benefits through Aetna Pharmacy® can also • View prescription details • Get cost estimates for brand name or generic drugs • Search for a pharmacy • Refill mail-order prescriptions *available for members in eligible plans

Report Generated by - MobSF v3.6.2 Beta

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

© 2023 Mobile Security Framework - MobSF | Ajin Abraham | OpenSecurity.

6.2 MobSF Analysis Report iOS Aetna App

File Name: Aetna Health 5.23.0.ipa
Identifier: com.aetna.ngx
Scan Date: Feb. 17, 2023, 9:16 p.m.
App Security Score: 42/100 (MEDIUM RISK)
Grade: B

FINDINGS SEVERITY

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
2	1	1	1	1

FILE INFORMATION

File Name: Aetna Health 5.23.0.ipa
Size: 89.17MB
MD5: 68052e613c4ffac2ff7e947fcef6666c
SHA1: 8a698c3674535da7a7afa723b0603bbeeb0981c8
SHA256: 3d1483598ef980dcf28a791f0641a80cde3c5987a4052b96783d1e4bf91860c

APP INFORMATION

App Name: Aetna Health
App Type: Swift
Identifier: com.aetna.ngx
SDK Name: iphoneos16.2
Version: 5.23.0
Build: 114239
Platform Version: 16.2
Min OS Version: 15.0
Supported Platforms: iPhoneOS,

BINARY INFORMATION

Arch: ARM64
Sub Arch: CPU_SUBTYPE_ARM64_ALL
Bit: 64-bit
Endian: <

#CUSTOM URL SCHEMES

URL NAME	SCHEMES
aetnadigital Editor	aetnadigital

APPLICATION PERMISSIONS

PERMISSIONS	STATUS	INFO	REASON IN MANIFEST
NSCalendarsUsageDescription	dangerous	Access Calendars.	Providing access to your calendar will allow us to automatically add scheduled consults at your request.
NSCameraUsageDescription	dangerous	Access the Camera.	This lets you take pictures for claims, video for virtual consultations and more.
NSFaceIDUsageDescription	normal	Access the ability to authenticate with Face ID.	Enabling Face ID allows you quick and secure access to your account.
NSHealthShareUsageDescription	dangerous	Read Health Data.	If you sync your steps and exercise info through HealthKit you can get even better support in care management and wellness programs.
NSHealthUpdateUsageDescription	dangerous	Write Health Data.	If you sync your steps and exercise info through HealthKit you can get even better support in care management and wellness programs.
NSLocationWhenInUseUsageDescription	dangerous	Access location information when app is in the foreground.	This will allow us to find the best Doctors, Urgent Care Centers, and Hospitals near you.
NSMicrophoneUsageDescription	dangerous	Access microphone.	Allow Aetna Health to access your microphone for virtual consultations with a health care provider.
NSPhotoLibraryUsageDescription	dangerous	Access the user's photo library.	Allow Aetna Health to access your photo library for virtual consultations with a health care provider.
NSSpeechRecognitionUsageDescription	normal	Allow app to send user data to Apple's speech recognition servers.	Speech recognition access will allow our Teladoc Health Assistant to help you with your needs.

PROTECTION	STATUS	SEVERITY	DESCRIPTION
NX	True	info	The binary has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.
PIE	True	info	The binary is build with -fPIC flag which enables Position Independent code. This makes Return Oriented Programming (ROP) attacks much more difficult to execute reliably.
STACK CANARY	True	info	This binary has a stack canary value added to the stack so that it will be overwritten by a stack buffer that overflows the return address. This allows detection of overflows by verifying the integrity of the canary before function return.
ARC	True	info	The binary is compiled with Automatic Reference Counting (ARC) flag. ARC is a compiler feature that provides automatic memory management of Objective-C objects and is an exploit mitigation mechanism against memory corruption vulnerabilities.
RPATH	True	warning	The binary has Runpath Search Path (@rpath) set. In certain cases an attacker can abuse this feature to run arbitrary executable for code execution and privilege escalation. Remove the compiler option -rpath to remove @rpath.
CODE SIGNATURE	True	info	This binary has a code signature.
ENCRYPTED	True	info	This binary is encrypted.
SYMBOLS STRIPPED	True	info	Symbols are stripped

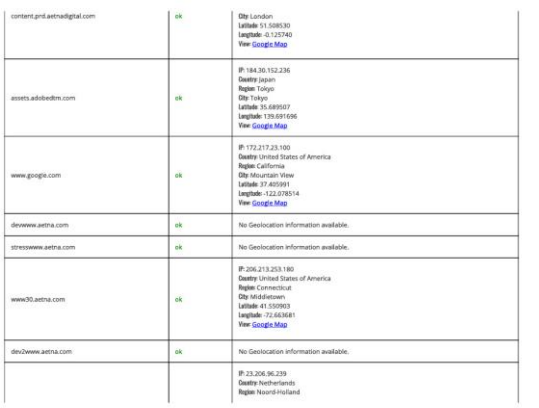
IPA BINARY CODE ANALYSIS

NO	ISSUE	SEVERITY	STANDARDS	DESCRIPTION
1	Binary makes use of insecure API(s)	high	CWE: CWE-676: Use of Potentially Dangerous Function OWASP Top 10: M7: Client Code Quality OWASP MASVS: MSTG-CODE-8	The binary may contain the following insecure API(s) _memcpy , _strlen , _fopen , _sscanf
2	Binary makes use of Logging function	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	The binary may use _NSLog function for logging.
3	Binary makes use of malloc function	high	CWE: CWE-789: Uncontrolled Memory Allocation OWASP Top 10: M7: Client Code Quality OWASP MASVS: MSTG-CODE-8	The binary may use _malloc function instead of calloc

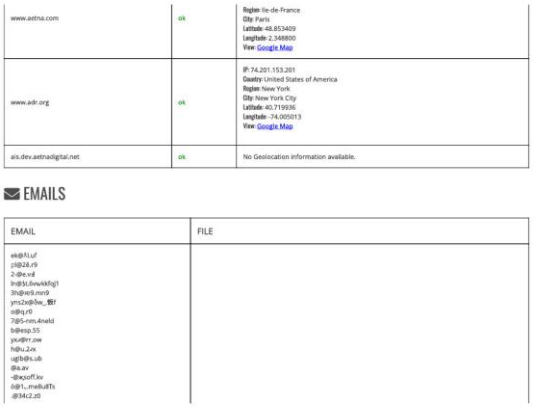
DOMAIN MALWARE CHECK

DOMAIN	STATUS	GEOLOCATION
		IP: 52.222.236.92 Country: United Kingdom of Great Britain and Northern Ireland Region: England

IPA BINARY ANALYSIS



content.prd.aetnadigital.com	ok	City: London Latitude: 51.508530 Longitude: -0.125740 View: Google Map
assets.adobedtm.com	ok	IP: 184.30.152.236 Country: Japan Region: Tokyo City: Tokyo Latitude: 35.689507 Longitude: 139.691696 View: Google Map
www.google.com	ok	IP: 172.217.23.100 Country: United States of America Region: California City: Mountain View Latitude: 37.405991 Longitude: -122.078514 View: Google Map
devwww.aetna.com	ok	No Geolocation information available.
stresswww.aetna.com	ok	No Geolocation information available.
www30.aetna.com	ok	IP: 206.213.253.180 Country: United States of America Region: Connecticut City: Middletown Latitude: 41.550903 Longitude: -72.663681 View: Google Map
dev2www.aetna.com	ok	No Geolocation information available.
		IP: 23.206.96.239 Country: Netherlands Region: Noord-Holland



www.aetna.com	ok	Region: Ile-de-France City: Paris Latitude: 48.853409 Longitude: 2.348800 View: Google Map
www.adr.org	ok	IP: 74.201.153.201 Country: United States of America Region: New York City: New York City Latitude: 40.719936 Longitude: -74.005013 View: Google Map
ais.dev.aetnadigital.net	ok	No Geolocation information available.

EMAIL	FILE
iOmxom-c9@mg.alli	
sj@s.g7d	
v6de@d.g83	
r@w.z2	
pa7@8ib.49	
e_@b.gn	
5@8.hsag	
9@q1ttoi.se	

EMAIL	FILE
ek@A1.uf	
p3@Z8.r9	
2-@e.xvl	
lrd@3.l.6vwkkfg1	
3k@et9.mn9	
yns2xd@w_.f6f	
o@q.r9	
7@5-rm.4neld	
b@esp.55	
yxu@r.ow	
h@u.2vs	
ugfb@s.ub	
@a.av	
-@xsoff.kv	
b@1-.mellu8Ts	
.@34c2.z0	

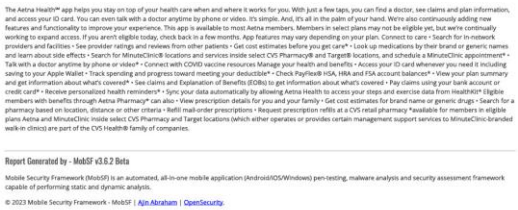
APP STORE INFORMATION

Title: Aetna Health

Score: 4.65062 **Features:** **Price:** 0.0 **Category:** Health & Fitness, Medical,
App Store URL: com.aetna.rxgx

Developer: Aetna Life Insurance Company
Developer ID: 380845819
Developer Website: http://www.aetna.com
Developer URL: https://apps.apple.com/us/developer/aetna-life-insurance-company/id380845819?uo=4
Supported Devices: iPhone5s-iPhone5s, iPadAir-iPadAir, iPadAirCellular-iPadAirCellular, iPadMiniRetina-iPadMiniRetina, iPadMiniRetinaCellular-iPadMiniRetinaCellular, iPhone6-iPhone6, iPhone6Plus-iPhone6Plus, iPadAir2-iPadAir2, iPadAir2Cellular-iPadAir2Cellular, iPadMini3-iPadMini3, iPadMini3Cellular-iPadMini3Cellular, iPodTouchSixthGen-iPodTouchSixthGen, iPhone6s-iPhone6s, iPhone6sPlus-iPhone6sPlus, iPadMini4-iPadMini4, iPadMini4Cellular-iPadMini4Cellular, iPadPro-iPadPro, iPadProCellular-iPadProCellular, iPadPro97-iPadPro97, iPadPro97Cellular-iPadPro97Cellular, iPhoneSE-iPhoneSE, iPhone7-iPhone7, iPhone7Plus-iPhone7Plus, iPad611-iPad611, iPad612-iPad612, iPad71-iPad71, iPad72-iPad72, iPad73-iPad73, iPad74-iPad74, iPhone8-iPhone8, iPhone8Plus-iPhone8Plus, iPhoneX-iPhoneX, iPad75-iPad75, iPad76-iPad76, iPhoneXS-iPhoneXS, iPhoneXSMax-iPhoneXSMax, iPhoneXR-iPhoneXR, iPad812-iPad812, iPad834-iPad834, iPad856-iPad856, iPad878-iPad878, iPadMini5-iPadMini5, iPadMini5Cellular-iPadMini5Cellular, iPadAir3-iPadAir3, iPadAir3Cellular-iPadAir3Cellular, iPodTouch7GenGen-iPodTouch7GenGen, iPhone11-iPhone11, iPhone11Pro-iPhone11Pro, iPadSeventhGen-iPadSeventhGen, iPadSeventhGenCellular-iPadSeventhGenCellular, iPhone11ProMax-iPhone11ProMax, iPhoneSESecondGen-iPhoneSESecondGen, iPadProSecondGen-iPadProSecondGen, iPadProSecondGenCellular-iPadProSecondGenCellular, iPadProFourthGen-iPadProFourthGen, iPadProFourthGenCellular-iPadProFourthGenCellular, iPhone12Mini-iPhone12Mini, iPhone12-iPhone12, iPhone12Pro-iPhone12Pro, iPhone12ProMax-iPhone12ProMax, iPadAir4-iPadAir4, iPadAir4Cellular-iPadAir4Cellular, iPadEighthGen-iPadEighthGen, iPadEighthGenCellular-iPadEighthGenCellular, iPadProThirdGen-iPadProThirdGen, iPadProThirdGenCellular-iPadProThirdGenCellular, iPadProFifthGen-iPadProFifthGen, iPadProFifthGenCellular-iPadProFifthGenCellular, iPhone13Pro-iPhone13Pro, iPhone13ProMax-iPhone13ProMax, iPhone13Mini-iPhone13Mini, iPhone13-iPhone13, iPadMiniSixthGen-iPadMiniSixthGen, iPadMiniSixthGenCellular-iPadMiniSixthGenCellular, iPadNinthGen-iPadNinthGen, iPadNinthGenCellular-iPadNinthGenCellular, iPhoneSEThirdGen-iPhoneSEThirdGen, iPadAirFifthGen-iPadAirFifthGen, iPadAirFifthGenCellular-iPadAirFifthGenCellular, iPhone14-iPhone14, iPhone14Plus-iPhone14Plus, iPhone14Pro-iPhone14Pro, iPhone14ProMax-iPhone14ProMax, iPadTenthGen-iPadTenthGen, iPadTenthGenCellular-iPadTenthGenCellular, iPadPro11FourthGen-iPadPro11FourthGen, iPadPro11FourthGenCellular-iPadPro11FourthGenCellular, iPadProSixthGen-iPadProSixthGen, iPadProSixthGenCellular-iPadProSixthGenCellular,

Description:



The Aetna Health℠ app helps you stay on top of your health care when and where it works for you. With just a few taps, you can find a doctor, see claims and plan information, and access your ID card. You can even talk with a doctor anytime by phone or video. It's simple. And, it's all in the palm of your hand. We're also continuously adding new features and functionality to improve your experience. This app is available to most Aetna members. Members in select plans may not be eligible yet, but we're continually working to expand access. If you aren't eligible today, check back in a few months. App features may vary depending on your plan. Connect to care • Search for in-network providers and facilities • See provider ratings and reviews from other patients • Get cost estimates before you get care* • Look up medications by their brand or generic names and learn about side effects • Search for MinuteClinic® locations and services inside select CVS Pharmacy® and Target® locations, and schedule a MinuteClinic appointment* • Talk with a doctor anytime by phone or video* • Connect with COVID vaccine resources Manage your health and benefits • Access your ID card whenever you need it including saving to your Apple Wallet • Track spending and progress toward meeting your deductible* • Check PayFlex® HSA, HRA and FSA account balances* • View your plan summary and get information about what's covered* • See claims and Explanation of Benefits (EOBs) to get information about what's covered • Pay claims using your bank account or credit card* • Receive personalized health reminders* • Sync your data automatically by allowing Aetna Health to access your steps and exercise data from HealthKit* Eligible members with benefits through Aetna Pharmacy* can also • View prescription details for you and your family • Get cost estimates for brand name or generic drugs • Search for a pharmacy based on location, distance or other criteria • Refill mail-order prescriptions • Request prescription refills at a CVS retail pharmacy *available for members in eligible plans Aetna and MinuteClinic inside select CVS Pharmacy and Target locations (which either operates or provides certain management support services to MinuteClinic-branded walk-in clinics) are part of the CVS Health® family of companies.

Report Generated by - MobSF v3.6.2 Beta

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

© 2023 Mobile Security Framework - MobSF | Ajin Abraham | OpenSecurity.

7 References

[1] Kønig, C., et al. (2014). The Mobile Security Landscape. Journal of Computer Security, 22(6), pp. 761-792.

[2] Joseph Chan Joo Keng, Tan Kiat Wee, Lingxiao Jiang, and Rajesh Krishna Balan, The Case for Mobile Forensics of Private Data Leaks:Towards Large-Scale User-Oriented Privacy Protection, School of Information Systems, Singapore Management University 2012.

[3] Hassanien, A. E., Haqiq, A., Tonellato, P. J., Bellatreche, L., Goundar, S., Azar, A. T., … Bouzidi, D. (Eds.). (2021). Proceedings of the International Conference on Artificial Intelligence and Computer Vision (AICV2021). Advances in Intelligent Systems and Computing.

[4] A. P. Felt, D. Wagner, Phishing on mobile devices, na, 2011.

[5] Bojjagani, S., Brabin, D. R. D., & Rao, P. V. V. (2020). PhishPreventer: A Secure Authentication Protocol for Prevention of Phishing Attacks in Mobile Environment with Formal Verification. Procedia Computer Science, 171, 1110–1119.

[6] Kim, K.; Shin, Y.; Lee, J.; Lee, K. Automatically Attributing Mobile Threat Actors by Vectorized ATT&CK Matrix and Paired Indicator. Sensors 2021, 21, 6522. [Google Scholar]

[7] Zitouni, R., Agueh, M., Houngue, P., & Soude, H. (Eds.). (2020). e-Infrastructure and e-Services for Developing Countries. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering.

[8] Murat Yesilyurt, Yildiray Yalman, Security Threats on Mobile Devices and their Effects: Estimations for the Future, International Journal of Security and Its Applications Vol. 10, No. 2 (2016), pp.13-26

[9] Verma, H., et al. (2017). Mobile Apps and Identity Theft: An Exploratory Study. Proceedings of the 2017 IEEE International Conference on Computer and Information Technology (CIT), pp. 212-217.

[10] Yavuz, T., et al. (2015). An Analysis of Operating System Vulnerabilities in Mobile Devices. Proceedings of the 2015 International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp. 1-6.

[11] Shanmugam, K., et al. (2016). Investigating the Security of Mobile Health Applications. Proceedings of the 2016 IEEE International Conference on Communications and Signal Processing (ICCSP), pp. 558-562.

[12] Khan, M. A., & Babar, M. A. (2017). Mobile Application Security: A Survey. IEEE Access, 5, pp. 8714-8734.

[13] Wang, X., et al. (2015). A Survey of Android Security Threats and Defenses. Proceedings of the 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), pp. 1910-1915.

[14] Liao, X., et al. (2016). Protecting Sensitive Data in Untrusted Android Environments Using Trusted Execution Environments. IEEE Transactions on Information Forensics and Security, 11(8), pp. 1795-1807.

[15] Schütte, J., & Bodden, E. (2015). Detecting Jailbroken iOS Devices. Proceedings of the 2015 IEEE International Conference on Mobile Services (MS), pp. 47-54.

[16] Wang, H., & Ou, X. (2016). A Survey on Reverse Engineering of Android Applications. IEEE Transactions on Software Engineering, 42(7), pp. 641-660.

[17] Morgan, T. D., et al. (2015). Hooking Detection Techniques for Mobile Applications. IEEE Transactions on Mobile Computing, 14(10), pp. 2049-2063.

[18] Zhang, J., & Gui, X. (2017). A Survey of Hooking Techniques and Their Applications. Journal of Computer Science and Technology, 32(3), pp. 415-429.

[19] Tajalizadehkhoob, S., et al. (2018). Debugger Detection Techniques for Android Applications. Journal of Computer Virology and Hacking Techniques, 14(4), pp. 259-273.

[20] Sadeghi, A., et al. (2016). Detecting Screen Capture-based Attacks on Android Devices Using Motion Sensors. Proceedings of the 2016 ACM Conference on Computer and Communications Security, pp. 1315-1326.

[21] Bejleri, A., et al. (2016). Security Analysis of iOS Password Vaults. Proceedings of the 2016 ACM Conference on Computer and Communications Security, pp. 1241-1252.

[22] Qasem, S., et al. (2017). Encryption in Android Applications: A Comparative Study. Proceedings of the 2017 IEEE International Conference on Cloud Computing and Big Data Analysis, pp. 29-34.

[23] Smith, M., and Small, S. (2014). Analyzing the Security of Third-Party iOS Applications. Proceedings of the 2014 USENIX Security Symposium, pp. 323-338.

[24] Jain, S., and Kumar, P. (2018). Analysis of Insecure Communication Channels in Mobile Applications. Proceedings of the 2018 International Conference on Computing, Communication, and Automation, pp. 126-130.

[25] Alshehri, S. A. E., et al. (2018). Mobile Application Security and Social Engineering Attacks: A Review. Proceedings of the 2018 International Conference on Innovations in Information Technology, pp. 58-63.

[26] Liu, J., et al. (2017). Security Analysis of Side-Loading on Android Devices. Proceedings of the 2017 IEEE Conference on Communications and Network Security, pp. 1-9.

[27] Wang, S., et al. (2018). A Survey on Mobile Man-in-the-Middle Attacks. IEEE Communications Surveys & Tutorials, 20(4), 3453-3473.

[28] Alshehri, A., et al. (2018). Detecting and Mitigating Flooding Attacks on Mobile Devices. Proceedings of the 2018 IEEE International Conference on Cybersecurity and Privacy, pp. 117-124.

[29] Alzahrani, A., et al. (2016). A Study on the Prevalence of Malware in Android Applications. Proceedings of the 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications, pp. 1-6.

[30] Islam, M. M., et al. (2015). Security Vulnerabilities in Mobile Communication Protocols. International Journal of Computer Networks and Communications, Vol. 7, No. 1, pp. 27-38.

[31] Khelifi, A., et al. (2019). Security Analysis of Mobile Applications and Mitigation Strategies Against Network Spoofing Attacks. International Journal of Network Security, Vol. 21, No. 6, pp. 1072-1082.

[32] J. Alnabulsi, S. Khan, and I. A. Khan, "Weak Password Analysis of Mobile Users," 2020 IEEE International Conference on Innovations in Information Technology (IIT), 2020, pp. 1-6.

[33] Goyal, S., & Singh, G. (2018). Risks of Using Mobile Applications on Unsecured Wi-Fi Networks. International Journal of Engineering and Technology (UAE), 7(2.26), 139-144.

[34] S. Ryu, Y. Jang, and H. J. Kim, "A Study on the Risk of Using Outdated Applications in Mobile Devices," 2017 4th International Conference on Information Science and Control Engineering (ICISCE), 2017, pp. 329-333.

[35] https://www.android.com/play-protect/

[36] https://developer.android.com/topic/security/sandboxing

[37] https://developer.android.com/guide/topics/permissions/overview

[38] https://source.android.com/security/verifiedboot/

[39] https://searchsecurity.techtarget.com/definition/verified-boot

[40] https://source.android.com/security/data-encryption

[41] https://support.google.com/googleplay/android-developer/answer/9859654?hl=en

[42] https://source.android.com/security/bulletin

[43] https://developer.android.com/topic/google-play-protect/treble

[44] https://developer.apple.com/app-store/review/

[45] https://support.apple.com/en-us/HT208108

[46] https://support.apple.com/en-us/HT204587

[47] https://www.apple.com/ios/security/

[48] Direito, A., Dale, L. P., Shields, E., Dobson, R., & Whittaker, R. (2014). Effectiveness of eHealth interventions for promoting physical activity in older adults: A systematic review. Journal of Medical Internet Research, 16(7), e161. doi: 10.2196/jmir.3050

[49] Bardus, M., Smith, J. R., Samaha, L., & Abraham, C. (2016). Mobile and web-based apps that support self-management and self-monitoring of dietary intake: A systematic review. Journal of Medical Internet Research, 18(6), e164. doi: 10.2196/jmir.5590

[50] Huberty, J., Eckert, R., Larkey, L., Kurka, J., de Moor, C., & Klemann, N. (2019). Smartphone-based meditation for myeloproliferative neoplasm patients: Feasibility study to inform future trials. Journal of Medical Internet Research, 21(4), e13150. doi: 10.2196/13150

[51] Firth, J., Torous, J., Nicholas, J., Carney, R., Pratap, A., Rosenbaum, S., ... & Sarris, J. (2017). The efficacy of smartphone-based mental health interventions for depressive symptoms: A meta-analysis of randomized controlled trials. World Psychiatry, 16(3), 287-298. doi: 10.1002/wps.20472

[52] Cook, D. A., Wittich, C. M., Daniels, W. L., West, C. P., Harris, A. M., Beebe, T. J., & Hartzband, P. I. (2018). Incentivizing the use of smartphone-based diagnostic apps: A randomized trial. JAMA Internal Medicine, 178(3), 385-392. doi: 10.1001/jamainternmed.2017.7853

[53] Oakley, A., Patel, D., Gerhards, A., Murrell, D. F., & Sweeney, M. R. (2018). Accuracy of a smartphone app for triage of skin lesions based on machine learning algorithms. JAMA Dermatology, 154(4), 402-406. doi: 10.1001/jamadermatol.2017.5376

[54] Mehrotra, A., Jena, A. B., Busch, A. B., & Souza, J. (2013). Telemedicine and ambulatory subspecialty care utilization among Medicare beneficiaries. JAMA, 309(22), 2437-2438. doi: 10.1001/jama.2013.5928

[55] Cook, D. A., Triola, M. M., & Ioannidis, J. P. (2014). More than meets the eye: Bias that lurks beneath the surface in visual grading studies. JAMA, 311(21), 2161-2162. doi: 10.1001/jama

[56] Ammenwerth, E., Schnell-Inderst, P., & Hoerbst, A. (2012). The impact of electronic patient portals on patient care: A systematic review of controlled trials. Journal of Medical Internet Research, 14(6), e162. doi: 10.2196/jmir.2238

[57] Lau, A. Y., Arguel, A., Dennis, S., Liaw, S. T., & Coiera, E. (2019). “Why didn't it work?” Lessons from a randomized controlled trial of a web-based personally controlled health management system for adults with asthma. Journal of Medical Internet Research, 21(4), e12380. doi: 10.2196/12380

[58] Jakicic, J. M., Davis, K. K., Rogers, R. J., King, W. C., Marcus, M. D., Helsel, D., ... Belle, S. H. (2016). Effect of wearable technology combined with a lifestyle intervention on long-term weight loss: The IDEA randomized clinical trial. JAMA, 316(11), 1161-1171. doi: 10.1001/jama.2016.12858

[59] Isaac, T., Zheng, J., & Jha, A. (2012). Use of UpToDate and outcomes in US hospitals. Journal of Hospital Medicine, 7(2), 85-90. doi: 10.1002/jhm.944

[60] Patterson, J. W., Mann, T., & Kirsner, R. S. (2011). Skin diseases: Diagnosis and management – an online medical education and clinical decision support tool. Journal of Drugs in Dermatology, 10(5), 569-575.

[61] Whitten, P., Holtz, B., & Laplante, C. (2017). Telemedicine adoption by different groups of physicians. Health Affairs, 36(12), 2268-2276. doi: 10.1377/hlthaff.2017.1012

[62] Chen, R., Powell, J., Reeves, D., Bardsley, M., & Majeed, A. (2019). Using a patient appointment booking website (Zocdoc) in NHS general

[63] Abraham MB, Nicholas JA, Smith GJ, et al. Use of a predictive algorithm in a closed-loop artificial pancreas system. Diabetes Care. 2013;36(9):2633–2640.

[64] Hua R, Tang Y, Li K, et al. A review of wearable sensor-based systems for real-time health monitoring. J Sens. 2021;2021:6565869.

[65] Jia J, Lu S, Zhang C, et al. A cloud-based 3D printing service platform for customized medical devices. 3D Print Med. 2021;7(1):4.

[66] Hollander JE, Carr BG. Virtually perfect? Telemedicine for Covid-19. N Engl J Med. 2020;382(18):1679-1681.

[67] Memic A, Tadayon M, Navarro M, et al. Soft robotics and microfluidics for minimally invasive surgery. Adv Healthc Mater. 2021;10(1):e2000798.

[68] Wang Z, Tang F, Wang J, et al. Wearable health sensors in personalized medicine: a systematic review of the literature. Am J Med Res. 2021;8(1):1-9.

[69] Martínez-Marín JL, Gil-López S, Sánchez-González Á, et al. A review of 3D printing mobile apps. Addit Manuf. 2021;40:101926.

[70] Janaka Senanayake, Harsha Kalutarage, Mhd Omar Al-Kadri, Andrei Petrovski, and Luca Piras. Developing secured android applications by mitigating code vulnerabilities with machine learning. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, pages 1255-1257, 2022

[71] Ashwag Albakri, Huda Fatima, Maram Mohammed, Aisha Ahmed, Aisha Ali, Asala Ali, and Nahla Mohammed Elzein. Survey on reverse-engineering tools for android mobile devices. Mathematical Problems in Engineering, 2022:1–7, 2022.

[72] Hilmi Abdullah and Subhi RM Zeebaree. Android mobile applications vulnerabilities and prevention methods: A review. 2021 2nd Information Technology To Enhance e-learning and Other Application (IT-ELA), pages 148–153, 2021.

[73] Juliza Mohamad Arif, Mohd Faizal Ab Razak, Sharfah Ratibah Tuan Mat, Suryanti Awang, Nor Syahidatul Nadiah Ismail, and Ahmad Firdaus. Android mobile malware detection using fuzzy ahp. Journal of Information Security and Applications, 61:102929, 2021.

[74] Gioacchino Tangari, Muhammad Ikram, I Wayan Budi Sentana, Kiran Ijaz, Mohamed Ali Kaafar, and Shlomo Berkovsky. Analyzing security issues of android mobile health and medical applications. Journal of the American Medical Informatics Association, 28(10):2074–2084, 2021.

[75] Rajindra Adhikari, Deborah Richards, and Karen Scott. Security and privacy issues related to the use of mobile health apps. ACIS, 2014.

[76] Achilleas Papageorgiou, Michael Strigkos, Eugenia Politou, Efthimios Alepis, Agusti Solanas, and Constantinos Patsakis. Security and privacy analysis of mobile health applications: the alarming state of practice. Ieee Access, 6:9390–9403, 2018.

[77] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and XiaoFeng Wang. Os-level side channels without procfs: Exploring cross-app information leakage on ios. In Proceedings of the Symposium on Network and Distributed System Security, 2018.

[78] https://developer.android.com/tools/adb

[79] https://frida.re/

[80] https://medium.com/@Kamal_S/mobile-security-framework-mobsf-setup-kali-linux-and-windows-afb055721c41

[81] https://github.com/MobSF/Mobile-Security-Framework-MobSF

[82] https://ibotpeaches.github.io/Apktool/

[83] https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jarsigner.html

[84] https://portswigger.net/burp