Penetration Testing and Privacy Assessment of Top-Ranked Health and Fitness Apps

An Empirical Study

Penetrationstestning och Integritetsbedömning av Toppklassade Hälso- och Fitnessappar

En Empirisk Studie

Albin Forsberg <forsberg.albin@telia.com>

Faculty of Health, Science and Technology

Master thesis in Computer Science

Second Cycle, 30 hp (ECTS)

Supervisor: Leonardo Iwaya, Karlstad University, Karlstad, SWE

Examiner: Tobias Pulls, Karlstad University, Karlstad, SWE

Karlstad, June 2nd, 2024

Abstract

Mobile health applications (mHealth apps), particularly in the health and fitness category, have experienced an increase in popularity due to their convenience and availability. However, this widespread adoption has raised concerns regarding the security and privacy of user data within these apps. This study investigates the security and privacy risks associated with ten top-ranked Android health and fitness apps, a set which accounts for 237 million downloads. By utilizing tools such as MobSF, Qualys SSL, and CLAUDETTE, we performed a static, dynamic, server-side, and privacy policy analysis in order to gain comprehensive insights into the security and privacy posture of the investigated mobile health and fitness apps. The results from the analysis revealed vulnerabilities in coding practices, hardcoded sensitive information, insecure encryption configurations, misconfiguration, and extensive domain communication. For instance, our analysis revealed that all apps stored their database API key directly in the code, with eight apps additionally exposing the database URL. Furthermore, six apps employed insecure encryption methods, such as CBC mode with PKCS5/PKCS7 padding (five apps) and ECB mode (two apps). In total, the apps interacted with 404 distinct domains. Notably, two apps communicated with more than 230 domains each, while a third app connected with over 100 domains. Despite these findings, developers demonstrated improved awareness and proficiency in addressing privacy and security risks compared to previous studies in the field. The study underscores the importance of continuous research to comprehensively understand the security and privacy landscape of health and fitness apps.

Keywords

Security, Privacy, Pentesting, Cyber Security, Vulnerability Assessment, Testing, Mobile Health Applications, Vetting, Health and Fitness Applications

Sammanfattning

Mobila hälsoapplikationer, särskilt de inom hälso- och fitnesskategorin, har ökat i popularitet på grund av deras bekvämlighet och tillgänglighet. Denna snabba ökning av användare har dock väckt oro för säkerheten och integriteten för användardata i dessa appar. I den här studien undersöks säkerhets- och integritetsriskerna i samband med tio topprankade Android-appar inom hälsa och fitnesskategorin, en uppsättning som står för 237 miljoner nedladdningar. Genom att använda verktyg som MobSF, Qualys SSL och CLAUDETTE, utförde vi en statisk, dynamisk, server och integritetspolicy analys för att få omfattande insikter kring säkerheten och integritetsläget för de undersökta apparna. Resultaten från analysen avslöjade sårbarheter i kodningsmetoder, hårdkodad känslig information, osäkra krypteringskonfigurationer, felkonfigurationer och omfattande domänkommunikation. Till exempel avslöjade vår analys att alla appar lagrade sin databas API-nyckel direkt i koden, varav åtta appar dessutom exponerade databasens URL. Dessutom använde sig sex appar av osäkra krypteringsmetoder, såsom CBC-läge med PKCS5/PKCS7-utfyllnad (fem appar) och ECB-läge (två appar). Totalt kommunicerade apparna med 404 unika domäner. Noterbart är att två appar kommunicerade med mer än 230 domäner vardera, medan en tredje app kommunicerade med över 100 domäner. Trots dessa fynd uppvisade utvecklare förbättrad medvetenhet och skicklighet i att hantera integritets- och säkerhetsrisker jämfört med tidigare studier inom området. Studien understryker vikten av kontinuerlig forskning för att heltäckande förstå säkerheten och integritetslandskapet för hälso- och fitnessappar.

Nyckelord

Säkerhet, Integritet, Penetrationstestning, Cybersäkerhet, Sårbarhetsbedömning, Testning, Mobila Hälsoapplikationer, Vetting, Hälso- och Fitnessapplikationer

Acknowledgements

I would like to thank my supervisor at Karlstad University, Leonardo Iwaya, for his support, motivation, and engagement, which has made this thesis project an enriching experience. His mentorship and insightful feedback have significantly contributed to the successful completion of this thesis project.

Acronyms

CPU	Central Processing Unit
URL	Uniform Resource Locator
IP	Internet Protocol
RAM	Random Access Memory
EU	European Union
APK	Android Application Package
HTTPS	HyperText Transfer Protocol Secure
HTTP	HyperText Transfer Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
AES	Advanced Encryption Standard
ECB	Electronic Code Book
MD5	Message Digest
GDPR	General Data Protection Regulation
IVs	Initialization Vector
ICC	Inter-Component Communication
API	Application Programming Interface
SHA1	Secure Hash Algorithm 1
DPOs	Data Protection Officers
DPIAs	Data Protection Impact Assessments
OFAC	Office of Foreign Assets Control
MobSF	Mobile Security Framework
OWASP	Open Web Application Security Project
DDoS	Distributed Denial of Service
MitM	Man in the Middle
CBC	Cipher Block Chaining
GCM	Galois/Counter Mode
PKIX	Public Key Infrastructure Using X.509 Certificates
mHealth	Mobile Health
PP	Privacy Policy
SA	Static Analysis
DA	Dynamic Analysis
SS	Server Side
RD	Responsible Disclosure

Contents

Chapter 1

Introduction

Mobile health applications (mHealth apps) have become increasingly popular due to their availability, convenience, and ability to provide accessible health-related solutions to users across multiple devices [24, 36]. While many subcategories of mHealth apps exist, some are more popular and widely used than others. Health and fitness is one of the most popular categories and has seen a massive increase in usage, especially between January 2019 and January 2020, when the downloads of the most popular health and fitness apps worldwide almost doubled from 8.84 million in 2019 to 16.28 million in 2020 [49].

Mobile health and fitness apps offer a wide range of functionalities, including fitness tracking, dietary management, and ways to find new exercises to improve your health at any time, anywhere. This was particularly useful during the COVID-19 pandemic, which forced gyms to close and people to stay at home, increasing the demand for alternative workouts at home. According to [51], the number of health and fitness apps available on the Google Play Store increased from around 30 thousand in Q1 2019 to approximately 60 thousand in Q4 2021. However, such a rapid increase in app production raises questions about the thoroughness of the app developers, especially regarding the security and privacy of the apps.

Previous studies examining the privacy and security posture of mHealth apps have found worrying results [25, 44], ranging from insecure communication, weak encryption, and access control to lacking, unfair, or unavailable privacy policies. These findings emphasize the need for further and continuous research to assess the extent and impact of these threats on user privacy and data security within mHealth apps. Such investigation would provide deeper insights into the prevalence and severity of these vulnerabilities, guiding the development of more effective mitigation strategies. For such reasons, we investigate the security and privacy risks of the subcategory mobile health and fitness of mHealth apps in this study.

1.1 Research Questions

To guide this thesis, we developed the following research questions (RQs):

- **RQ1**: *What are the security and privacy risks in top-ranked health and fitness apps?* **Objectives:** to identify the security and privacy risks specific to health and fitness apps; and, to conduct empirical studies to evaluate the extent to which these risks are present in existing health and fitness apps.

- **RQ2**: *How can these identified risks be mitigated?* **Objective:** to propose recommendations for improving the security and privacy of health and fitness apps.

1.2 Methodology

Briefly, the study's methodology involved three key stages: (i) app selection, (ii) privacy and security analysis, and (iii) responsible disclosure. The app selection process employed a Google Play scraper to filter apps available in four English-speaking countries, with a minimum rating of 4 stars and over 1 million downloads, thus focusing only on top-ranked apps. For the privacy and security analysis, various tools were used, including Mobile Security Framework (MobSF) for static and dynamic analysis, Qualys Secure Sockets Layer (SSL) for server-side assessment, and WebFX, CLAUDETTE, and GDPRWise for a privacy policy analysis. Additionally, a manual analysis of the tools' outputs was conducted throughout the analysis process. Lastly, the responsible disclosure process involved compiling all findings into a comprehensive report and communicating them to the respective app companies via email, including relevant files such as web traffic data. A detailed description of these methodologies and methods is presented in Chapter 3.

1.3 Contribution

This study contributes valuable insights into the current privacy and security posture of some of the most popular, top-ranked health and fitness apps from four English-speaking countries by empirically studying the apps and identifying vulnerabilities and their prevalence within the apps. Furthermore, this study also proposes recommendations for mitigation tactics for the identified vulnerabilities, aiming to assist developers and stakeholders in improving the security and privacy of their apps. Previous studies have investigated similar scenarios in different categories of mHealth apps, yet this study differs in scope, focus, and methods.

1.4 Limitations

While this study offers valuable insights into the security and privacy of health and fitness apps, several limitations should be acknowledged. Firstly, the study's scope was limited to only ten top-ranked apps from the Google Play Store because of time constraints. Additionally, reliance on single tools such as MobSF for analysis and Qualys SSL for server-side assessment may have overlooked nuanced security issues. Furthermore, while AI-driven tools like WebFX, CLAUDETTE, and GDPRWise were employed for privacy policy analysis, their accuracy limitations should be considered when interpreting the findings. A detailed presentation of these limitations is presented in Chapter 5.7.

1.5 Ethics

This study adheres to ethical hacking guidelines for security research to try to find and mitigate security and privacy vulnerabilities in the apps [7]. Prior to public disclosure, identified vulnerabilities were reported to the respective app companies or developers, allowing sufficient time for remediation efforts. Furthermore, all results presented in this study are de-identified to ensure that the focus remains on the security and privacy vulnerabilities rather than on specific companies. This approach also ensures that possible vulnerabilities reported in this study are not used to exploit the apps' or expose user data. Additionally, this project received ethical approval from the Ethical Advisor at Karlstad University. It is also worth noting that this study does not tamper with the application infrastructure nor capture or handle the personal data of any users other than the one created explicitly for specified tests.

1.6 Outline

This thesis is structured as follows: Chapter 2 provides a comprehensive background on the project, including an overview of mobile health and fitness apps, contextual factors driving the study, and a review of relevant literature and previous research. Chapter 3 details the methodology, outlining research design, data collection methods, and analytical techniques. In Chapter 4, we present the results of our study, highlighting security vulnerabilities and privacy concerns identified in the analyzed apps. Chapter 5 discusses the implications of our findings and addresses limitations. Finally, Chapter 6 concludes the thesis by summarizing key insights, reflecting on contributions, and suggesting possibilities for future research in the field of mobile health and fitness app security and privacy.

Chapter 2

Background

During the COVID-19 pandemic, the popularity of mobile health and fitness applications experienced an increase in response to the lockdown measures and the limited availability of physical activities [32]. Mobile health and fitness applications, a subcategory of mHealth apps, offer a wide range of functionalities, including fitness tracking, dietary management, mental health support, and telemedicine consultations. Examples of such apps include fitness trackers (e.g., Fitbit and Garmin), calorie counters (e.g., MyFitnessPal), and meditation apps (e.g., Headspace).

As the demand for mHealth apps continues to rise, the number of applications available on the iOS App Store and Google Play Store has increased from around 55,000 to more than 90,000 over seven years between 2015 and 2022 [50, 51]. According to data from [49], the download figures for leading fitness and workout mobile apps worldwide significantly increased from 2019 to 2020, nearly doubling from 8.84 million to 16.28 million. Remarkably, this upward trend has persisted, with the download numbers remaining consistently high since then. Even with a slight decline, the figures have stabilized at this elevated level, underscoring the popularity and widespread adoption of these applications among users.

However, due to the high prevalence of mHealth apps and the sensitivity of the health data they collect, strong privacy and security measures must be implemented to safeguard user's personal information and ensure compliance with data protection regulations. In Table 2.0.1, we provide an example of data that might be collected in mHealth apps.

Table 2.0.1: Categories of Personal Data.

Data Category	**Data**
Personal Basic Information	Name, birthday, gender, country, email
Health Information	Height, weight, medical information
Device Information	Phone number, location

2.1 OWASP Mobile Security

The OWASP foundation[1] is a nonprofit organization dedicated to improving software security. One of the key areas of focus for OWASP is mobile security, given the widespread use of mobile devices and the increasing number of mobile applications available in various app stores. OWASP supplies a top 10 list[2] containing the most critical security risks facing mobile applications which is periodically updated to reflect new emerging threats. The OWASP Mobile Top 10 provides developers, security professionals, and stakeholders with insights into common vulnerabilities and best mitigation practices.

The current top 10 list includes insufficient cryptography, insecure data storage, insecure communication, inadequate privacy controls, and more. These risks highlight the diverse range of security challenges facing mobile app developers and the importance of implementing robust security measures in mobile app development. It is essential for developers and stakeholders to stay informed about the latest security risks identified by OWASP and to incorporate best practices for mobile app security into their development processes. By prioritizing security, developers can help protect user data and safeguard sensitive information.

Given their relevance to the cybersecurity community, this study uses the OWASP guidelines to support our claims of the identified threats. This helps us highlight our findings' significance and relevance in the context of mobile health and fitness applications. It also supplies stakeholders with actionable insights to effectively address and mitigate potential security risks in their applications.

2.2 Privacy & Security Concepts

Privacy and security are two crucial concepts that play a vital role in safeguarding the sensitive data of individuals and organizations. While security primarily focuses on protecting data from unauthorized access, disclosure, and tampering, emphasizing the preservation of confidentiality, integrity, and availability [23], privacy emphasizes individuals' control over their personal information. Privacy encompasses broader principles such as transparency, intervenability, autonomy, integrity, and fairness, which ensure that individuals have clear and understandable information about how their data is collected, used, and shared [22].

2.2.1 Privacy

Privacy refers to an individual's right to manage their personal information, including its collection, use, and sharing [55]. It empowers individuals to keep some aspects of their lives private and have a say in how others handle their data.

[1] OWASP Website: `https://owasp.org/about/`

[2] Top 10 Mobile Risks - OWASP Mobile Top 10 2024 (`https://owasp.org/www-project-mobile-top-10/2023-risks/`)

In the context of information technology and data processing, privacy concerns arise when individuals' data is collected, stored, analyzed, or shared by organizations, governments, or other entities without their consent or knowledge. Personal data can include a wide range of information, such as names, addresses, phone numbers, email addresses, financial details, health records, and browsing histories [23].

Privacy protection involves implementing policies, practices, and technologies to ensure that individuals' personal information is handled in a manner that respects their rights and preferences. This includes obtaining informed consent before collecting data, minimizing the collection and retention of personal information, implementing security measures to protect data from unauthorized access or disclosure, and providing individuals with transparency and control over their data.

Privacy laws and regulations, such as the GDPR, aim to protect individuals' privacy rights by establishing rules and requirements for organizations that handle personal data. These laws often require organizations to inform individuals about their data processing practices, obtain consent for data collection and processing activities, and provide mechanisms for individuals to access, correct, or delete their personal information [21].

2.2.2 Security

Security refers to the measures taken to protect assets, such as data, systems, and resources, from unauthorized access, misuse, modification, or destruction. In the context of computer systems and networks, security involves safeguarding digital information and ensuring the confidentiality, integrity, and availability of data [28].

Confidentiality ensures that only authorized individuals or entities can access sensitive information, preventing unauthorized disclosure [5]. Integrity ensures that data remains accurate and unaltered, protecting it from unauthorized modification or tampering [5]. Availability ensures that information and resources are accessible to authorized users when needed, preventing service disruptions [5].

Security measures can include encryption to protect data in transit and at rest, authentication mechanisms to verify the identity of users, access controls to limit user permissions based on roles and privileges, and intrusion detection systems to monitor and respond to unauthorized access attempts. Security protocols and best practices are also essential for designing and implementing secure systems and networks [28].

2.3 Importance of Security & Privacy in mHealth Apps

As the popularity of health and fitness apps and trackers continues to increase, so does the importance of ensuring the security and privacy of these applications. With

users entrusting these apps with sensitive personal information, such as their exercise routines, dietary habits, height, and weight [17] (see Table 2.0.1), the need to safeguard this data against unauthorized access and misuse becomes extremely important.

Findings from earlier studies, including those by [17, 25, 26], have identified substantial vulnerabilities associated with the attack surfaces outlined in Table 2.3.1. Moreover, these studies have revealed shortcomings in the accessibility and comprehensibility of privacy policies. For instance, earlier studies found that some privacy policies were either not properly linked to their corresponding app stores [26] or were written in language that required a university level of education to comprehend them effectively [25].

These vulnerabilities emphasize the need for robust mitigation tactics to address privacy and security risks in mHealth apps. Developers are responsible for implementing effective measures to safeguard sensitive user data and protect against potential breaches. Given the known vulnerabilities and the potential consequences of data breaches, there is no excuse for developers to neglect the implementation of comprehensive security and privacy measures. Failure to mitigate these risks may lead to data leakage, potentially exposing sensitive user data to unauthorized access.

Table 2.3.1: Attack Surfaces in Android Applications [17].

Attack Surface	**Description**
Internet	Sensitive information is sent over the Internet with insecure protocols, e.g., HTTP.
Third party	Sensitive information is stored or shared with third parties.
Bluetooth	Sensitive information collected by Bluetooth-enabled health devices can be sniffed or injected.
Logging	Sensitive information is put into system logs or saved in log files that are not secured.
SD Card Storage	Sensitive information is stored as unencrypted files on the SD card, publicly accessible by any other app.
Exported Components	Android app components, intended to be private, are set as exported, making them accessible by other apps
Side Channel	Sensitive information can be inferred by a malicious app with side channels, e.g., network package size, sequence, timing, etc.

2.4 GDPR

In recent years, the advancements in digital technologies and the exponential growth of online data collection have raised significant concerns regarding protecting the individual's privacy and personal data. In response to these challenges, the European

Union (EU) introduced GDPR in May 2018 [8]. GDPR is a framework designed to harmonize data protection laws across EU member states and enhance the rights of individuals regarding the processing and privacy of their personal data.

The GDPR aims to empower individuals by granting them greater control over their data and imposing obligations on organizations that collect, process, and store their personal data. The regulation applies to all entities that handle the personal data of EU residents. Compliance of GDPR includes aspects such as data collection, storage, sharing, and disposal [8].

One of the key principles of GDPR is the concept of "data protection by design and by default" [20], which requires organizations to implement appropriate technical and organizational measures to ensure the security and privacy of personal data from the outset of the data processing life-cycle. This principle emphasizes the importance of integrating data protection considerations into the design and development of products, services, and business processes, thereby minimizing the risk of data breaches and privacy violations [20].

Furthermore, GDPR introduces several rights for individuals to exercise greater control over their data, including the right to access, rectify, and erase their data, as well as the right to data portability and the right to object to certain data processing activities. Requirements for greater transparency of data processing also empower individuals to make informed decisions about using their personal information and hold organizations accountable for their data processing practices [8, 20].

In addition to enhancing individual rights, GDPR imposes strict obligations on organizations, including data controllers and processors, to ensure compliance with the regulation. For instance, these obligations include implementing appropriate security measures to protect personal data [8], obtaining explicit consent for data processing activities [20], conducting Data Protection Impact Assessments (DPIAs) for high-risk processing activities [19], and appointing Data Protection Officers (DPOs) to oversee compliance efforts [19].

2.5 Ethical Hacking

Ethical hacking, also known as penetration testing or *"the practice of hacking without malicious intent"* [47], plays a vital role in defending the public interest, safeguarding digital assets, and identifying vulnerabilities before malicious actors can exploit them.

According to [7], there is virtually no difference between a malicious and an ethical hacker in the techniques used to identify vulnerabilities. Instead, what separates the two is the intention of a system breach. While a malicious hacker might use identified vulnerabilities for something mischievous, an ethical hacker will instead *"report back to owners with the vulnerabilities they found and instructions for how to remedy them"* [47].

By adopting ethical hacking methodologies, this study seeks to identify and address potential vulnerabilities within these applications, ultimately aiming to fortify their defenses against cyber threats and safeguard the sensitive information handled by the apps. It's also worth noting that this study does not tamper with the application infrastructure nor capture or handle the personal data of any users other than the one specifically created for this study.

2.6 Related Work

Several studies investigate the security and privacy concerns related to mHealth apps. These studies aim to understand the past and present state of security and privacy risks of such apps and find solutions to identify, prevent, and mitigate such risks.

In this section, we review some of the prominent studies that have explored various aspects of security and privacy in Android mHealth apps, as shown in Table 2.6.1. Each study presents valuable insights into the challenges, trends, and best practices for safeguarding user data and privacy within the mHealth app ecosystem. By combining the findings and approaches of these studies, we aim to pinpoint areas for further research in this rapidly developing field.

2.6.1 Security & Privacy in mHealth Apps

Many mHealth apps available today collect and handle sensitive user data without employing appropriate security measures [25, 52, 53]. Studies have shown that many mHealth apps rely on unencrypted communication, transmitting data without encryption, and sending sensitive user information over insecure channels [52, 53]. In addition, a majority of analyzed mHealth apps have been found to transmit sensitive data without encryption over the internet, using weak encryption and hashing methods such as Electronic Code Book (ECB) cipher mode and Message Digest (MD5) algorithm [35]. In fact, for many years, studies have highlighted issues such as the misuse of cryptographic APIs and insecure Initialization Vector (IVs) [10]. Recent research further emphasizes that insecure encryption practices remain prevalent in mHealth apps, with findings revealing the widespread use of vulnerable IVs and ciphers [25].

For instance, an empirical study [25] conducted penetration tests on 27 mHealth apps, revealing numerous security and privacy vulnerabilities. These included information disclosure, weak access control, and excessive data permissions, raising serious concerns about user data privacy. Similarly, another study [44] highlighted issues such as excessive permissions and unauthorized data transmission to third parties, with instances of location tracking without user consent. Even earlier studies, such as [26], have highlighted significant shortcomings in mHealth app security, finding that out of 154 mHealth apps studied, health data encryption was rarely provided [26].

Privacy policies, essential for transparency and legal compliance, often fall short in

Table 2.6.1: Comparison of the existing works on privacy and security for mobile health applications according to their scope of analysis.

Ref	Year	N. Apps	SA	DA	SS	RD	PP	Limitations
[52]	2021	20991	✓	✓			✓	(i) Limited analysis because of scope. (ii) Limited to automated tools.
[53]	2021	20991	✓	✓				(i) Limited to automated tools. (ii) Focus on security testing.
[25]	2023	27	✓	✓	✓	✓	✓	(i) Limited to top-ranked mental health apps. (ii) PPs analyzed using AI-assisted tools
[26]	2015	154	✓	✓	✓		✓	(i) No substantial results were presented. (ii) No manual verification of static analysis results.
[44]	2018	20	✓	✓	✓		✓	(i) Limited to apps focused on health conditions or diseases. (ii) Focus on data protection and safe communication.
[16]	2014	160	✓	✓				(i) Focus on non-permission threats.
[35]	2022	30	✓					(i) Limited to HIPAA compliance
[17]	2014	160	✓					(i) Focus on the security of communication
[18]	2018	100	✓	✓				(i) Limited to the security of user data.
[10]	2013	11748	✓					(i) Limited to static analysis. (ii) Does not propose improvements.
This study	2024	10	✓	✓	✓	✓	✓	(i) Limited to top ranked Android health & fitness apps. (ii) PPs analyzed using AI-assisted tools.

mHealth apps, intensifying the privacy risks. Research [52] found a significant portion of mHealth apps lacking valid privacy policies, while those with policies exhibited extreme variations in compliance behavior. Furthermore, analyses of privacy policies using tools like PrivacyCheck [25] revealed deficiencies in user control and GDPR compliance, compounded by the poor readability of policy language.

Although the scope of these studies varies, with some being outdated [10, 16, 17, 26], more recent research [25, 35, 52, 53] continues to reach similar conclusions using similar methods. These studies consistently find that the security and privacy of mHealth apps are inadequate, with insufficient measures in place to protect users' sensitive data.

2.6.2 Privacy-Focused Analysis

(In-)Secure Communication

The study [25], which aimed to identify and understand data privacy (and security) incorporated in mHealth Apps by studying 27 mental health apps in-depth, found multiple security and privacy threats by conducting a series of empirical tests. The findings were categorized using the LINDDUN threat taxonomy, leveraging it as a standardized terminology to communicate privacy risks in software applications. LINDDUN stands for Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, and Non-compliance. The study found that 18 apps disclosed log-in and contextual data in the network traffic, generally had weak access control, and had an average of 5.6 dangerous permissions (min 3 - max 30), granting access to unnecessary permissions beyond the scope of the app's intended use. It was also discovered that, on average, 4.1 of the apps' permissions were unnecessary and that the apps' data were shared with an average of 11.9 third party servers (min 1 - max 64) [25]. The study concluded that information disclosure threats were highly prevalent due to insecure programming, which left user data vulnerable. In addition, when this type of data is shared with third parties, it could facilitate the profiling of users for exploitative advertising [25], which creates threats related to linkability, identifiability, non-repudiation, and detectability.

Contributions from other researchers evaluating the privacy of mHealth Apps have come to similar conclusions and findings. For instance, [44] found that some applications requested permissions that stretched beyond the intended scope. These permissions included access to Bluetooth and the microphone without any apparent need. The conclusion drawn from the situation was that an ad library used Bluetooth permission to track users' location [44]. Notably, the same study also found that 35% (7/20) [44] of apps transmitted the postal address or geolocation to vendors or third parties, whereas three apps did so over HTTP.

Privacy-Policies

In addition to the technical evaluations carried out via penetration testing, the privacy policies of mHealth apps play a significant role in addressing security and privacy concerns. Privacy policies are essential for applications to provide transparency, ensure legal compliance, build user trust, manage risks, and meet user expectations regarding data privacy and security.

Unfortunately, many mHealth app developers seem to forget this, as it was found that 28.1% [52] from a total of 20,991 mHealth apps had no privacy policy text at all. Interestingly, the study noted that apps tended to exhibit extremes in compliance behavior, either fully complying with the privacy policy or not complying at all. Specifically, 34.0% of apps demonstrated full compliance, while 49.0% showed non-compliance, attributed to either the absence of a privacy policy (21.0%) or all user data transmissions violating the privacy policy (28.1%) [52]. However, it was also found that there was a correlation between the popularity of the application and the prevalence of privacy policies, where around 94.4% [52] out of 589 mHealth apps with over 1 million downloads had them included on Google Play.

However, in the case of privacy policies being present, [25] examined the privacy policies of 26 mHealth Apps using a tool called PrivacyCheck [56], which assesses and rates an app on two key criteria, user control over privacy and GDPR compliance. On average, the apps received a user control score of 59 out of 100 (standard deviation = 15.14) and a GDPR score of 63.1 out of 100 (standard deviation = 31.25). Furthermore, when considering the readability of the policies [25] found that according to the Flesch-Kincaid reading ease measurement, the majority of apps (89%, n = 27) [25] scored between 30-50 on the readability index. This indicates that their privacy policies are difficult to comprehend and require a college-level education.

2.6.3 Security-Focused Analysis

Encryption & Protocols

According to the findings of [53], which investigated the user data collection of 20,991 mHealth Apps, show that as much as 45% of the investigated apps relied on unencrypted communication and that 23% of the apps transmitted sensitive user information such as passwords and location information on insecure traffic. In line with other studies, [16] has also found that most mHealth apps transmit sensitive data without encryption over the internet. Of the 27 apps analyzed [16], 23 (85.2%) possessed internet access permission. Among these, 19 apps (70.4%) utilized internet permissions mainly for displaying advertisements, while eight apps (29.6%) used internet permissions for transmitting data over the internet [16].

Another study found that, out of 30 apps examined, a majority of the apps opted for weak cryptographic methods, such as the ECB mode and the MD5 hashing algorithm, for data encryption and decryption [35]. Moreover, the study also concluded

that the adoption of advanced security measures, such as Advanced Encryption Standard (AES) encryption, Encoding-Decoding, Realm Database Encryption, Public Key Infrastructure Using X.509 Certificates (PKIX) Algorithm, and FireBase Authentication, remains relatively low, with less than 20% [35] of the surveyed apps employing these techniques.

Developers who fail to prioritize security make even outdated studies relevant. In their work, [10] discusses the misuse of cryptographic APIs in Android applications. Their findings reveal that out of 11,748 apps, 65% (7,656) used the ECB mode for encryption. This is unsuitable as ECB mode is deterministic, meaning that identical plaintexts always produce identical ciphertexts, making patterns visible to an adversary. The study also found that 511 out of 11,748 apps used static or fixed IVs [10], which could be perhaps considered a "good statistic". However, recent findings by [25] indicate that insecure IVs are more prevalent in mHealth apps today. Of the 27 top-ranked mHealth apps analyzed, 12 used insecure IVs [25], which is concerning. This study also had findings similar to [35], finding that seven out of the 27 apps used insecure cryptographic functions such as MD5 or Secure Hash Algorithm 1 (SHA1), and one app using ECB mode.

Server-side

In addition to examining client-side security measures, studies have also conducted comprehensive analyses of SSL web servers using the Qualys SSL Labs tool [25, 44]. This tool conducts a series of tests on specified web servers, evaluating the validity and trustworthiness of certificates and inspecting server-side SSL configurations. The tool assesses protocol support, key-exchange mechanisms, and cipher support to gauge the overall security posture of the web server with a letter grade scale [3] (A+, A, B, C, D, E, F, T). According to the findings of [44], out of 117 HTTPS connections established with third party servers, 108 servers had a grade of C or higher, while six servers received an F and three servers received a T grade. Moreover, out of 11 HTTPS connections made to the app vendors, five were rated A, three were rated B, one was rated C, and two were rated T [44].

Authentication & Access Control

Android uses Inter-Component Communication (ICC) as its main method of communication between apps. ICC acts as a connection between different components within and between different apps. Robust access control mechanisms for ICC are vital for enforcing security and privacy policies to ensure that only authorized components or apps can communicate and access sensitive data. In a study conducted by [18], access control for 100 sampled apps was reinforced, resulting in the detection and prevention of all attacks, including data leakage, privilege escalation, and misuse of permissions [18]. Before the access control reinforcement, the attacks scored 73 out

[3]Further details on the Qualys SSL Server Rating Guide can be found at `https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide`

of 100 apps [18] with data leakage, 53 out of 100 apps [18] with the potential to perform privilege escalation attacks, and 55 out of 100 apps [18] had no protection or authentication. Similarly to [18], the results of [35] underscore that not all mHealth apps are devoid of issues. Specifically, out of the 30 apps examined in the study, only nine (30%) [35] were found to implement authorization mechanisms to access sensitive resources.

2.6.4 Previous Work vs This Study

Many existing studies either lack comprehensive coverage, fully rely on automated analysis methods, or have limited scopes that hinder their ability to provide actionable insights. For example, some studies focus solely on security testing without considering privacy implications. In contrast, others are restricted to specific subsets of mHealth apps or rely heavily on automated tools, known for high false-positive rates.

This study aims to identify and mitigate the security and privacy of Android health and fitness apps. To achieve this, a series of penetration tests are carried out, including static, dynamic, and server-side analysis, and a review of privacy policies. This study aims to corroborate findings and provide further insights into security and privacy in mHealth apps, focusing on an area not thoroughly investigated in the related work. A comparison of our study's contributions with previous research efforts can be found in Table 2.6.1.

Chapter 3

Methodology

This chapter presents the methodology employed to empirically assess the security and privacy risks associated with mHealth apps. The methods include static and dynamic analysis, server-side configuration analysis, and privacy policy analysis, utilizing established methods from previous literature as well as commonly used security tools for risk assessment. The chapter covers the data collection process, including the selection criteria for mHealth applications, data extraction techniques, and analysis frameworks used to identify possible risks.

Figure 3.0.1 provides a graphical overview of this study's fundamental methodologies, specific procedures, and tools – heavily inspired by previous work of [25].

3.1 App Selection Process

The app selection process aimed to identify the top-performing health and fitness applications across four English-speaking countries: Australia, Canada, the United States, and the United Kingdom. Utilizing the Google Play Scraper tool[1], the top 30 apps from each of these countries were retrieved and analyzed (see search script in Listing 3.1). Choosing English-speaking countries for app selection provided practical advantages in terms of accessibility, comprehensibility, and consistency, thereby enhancing the correctness and relevance of the study's findings.

Two main criteria were established to ensure the selection of high-quality applications. Firstly, only apps with a user rating of 4.0 or higher were considered. This criterion ensured that only well-received applications were included in the study. Additionally, apps with a significant user base, defined as having more than 1 million downloads, were prioritized, indicating widespread popularity and relevance among users.

Furthermore, only those present in all four countries were retained to ensure consistency and relevance across all selected apps. This criterion helped maintain a

[1]Google Play Scraper (https://github.com/JoMingyu/google-play-scraper)

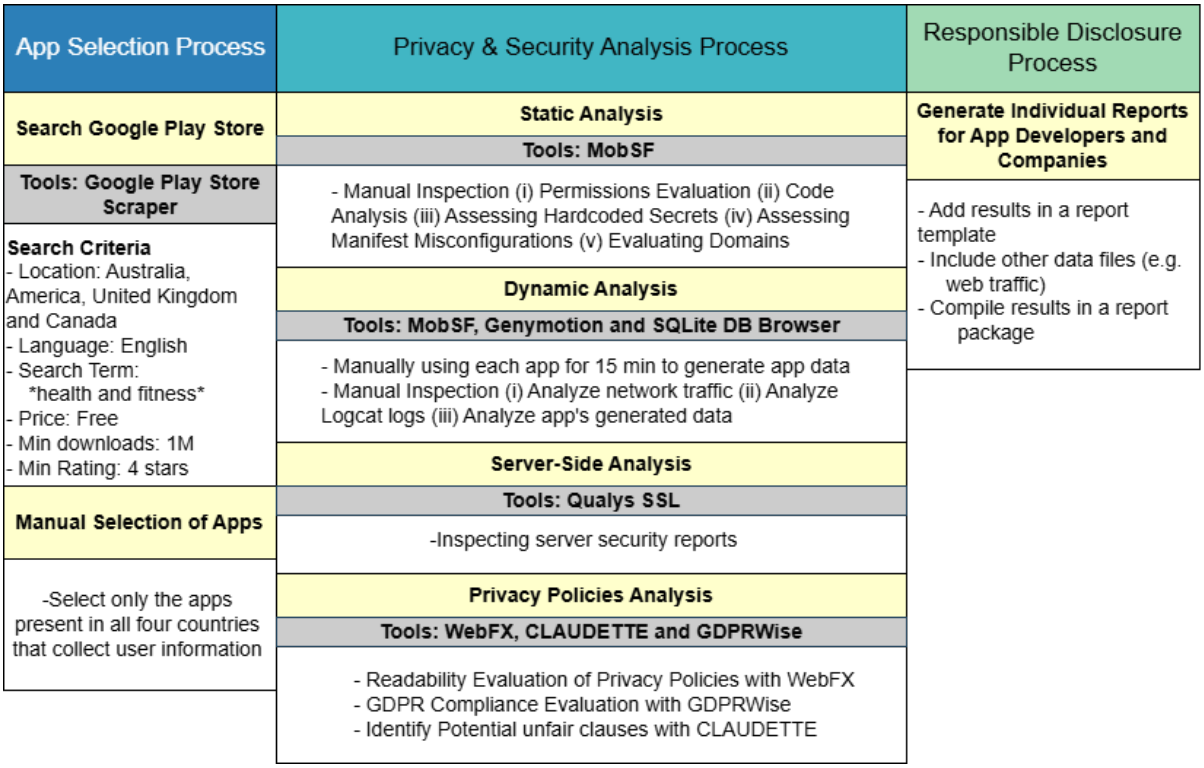

Figure 3.0.1: An overview of the methods used for investigating the privacy & security issues in health and fitness apps.

balanced representation across different regions and minimized biases associated with geographic variations in app popularity.

Finally, among the remaining applications, preference was given to those for which Android Application Package (APK) files were available on the APKMirror website[2]. This additional criterion facilitated access to the underlying application files, enabling static and dynamic analysis of the app.

Following the application of these selection criteria, the initial pool of 30 apps from each country was narrowed down to 15 apps based on the first three criteria. Subsequently, after applying the additional criterion related to the availability of APK files, the final selection concluded in 10 apps from the health and fitness category. All chosen apps were required to collect some form of personal or sensitive data from users, ensuring that the selected applications were directly relevant to the study's focus on privacy and security assessment of mHealth apps.

```
from google_play_scraper import search
import pandas as pd
country_codes = ["us", "uk", "au", "ca"]


for country_code in country_codes:
    result = search(
        "health and fitness",
```

[2]APKMirror (https://www.apkmirror.com/)

```
        lang="en",
        country=country_code,
        n_hits=30

    )
    if result:
        df = pd.DataFrame(result)
        filename = f"google_play_apps_{country_code}.xlsx"
        df.to_excel(filename, index=False)
        print(f"Saved data for {country_code} to {filename}")
    else:
        print(f"No results found for {country_code}")
```

Listing 3.1: Google Play Scraper search script.

3.2 Privacy & Security Analysis Process

In this section, the Privacy and Security Analysis Process employed to examine the integrity of health and fitness applications is described. The approach is similar to that of [25] yet differs in the scope, focus and tools (read more in section 2.6). The approach is based on previous work and is tailored to uncover vulnerabilities, inconsistencies, and other threats related to Android applications.

The analysis begins with a static analysis utilizing MobSF (see section below), complemented by manual inspection. This involves evaluating permissions, code, hardcoded secrets, manifest configurations, and domain assessments to identify potential security loopholes. Following that, a dynamic analysis is conducted utilizing MobSF and SQLite Browser, alongside a manual inspection that analyses network traffic, Logcat logs, and app-generated data to uncover potential vulnerabilities.

Simultaneously, a server-side analysis is performed on the discovered communicated domains using Qualys SSL to evaluate server security levels of HTTPS data transmissions. Additionally, privacy policies associated with the apps are evaluated for readability with WebFX, GDPR compliance with GDPRWise, and identification of potential unfair clauses with CLAUDETTE.

MobSF

MobSF[3], is an open-source automated mobile application security testing tool. It is designed to help developers, security researchers, and penetration testers assess the security posture of Android and iOS mobile applications. MobSF offers a wide range of features and capabilities, including Static and dynamic analysis. After analyzing an application, MobSF generates comprehensive reports detailing the identified security issues. When possible MobSF also provides links to the specific location of the detected issue, enabling developers to assess the situation quickly.

[3]MobSF website: `https://github.com/MobSF/Mobile-Security-Framework-MobSF`

3.2.1 Static Security Analysis

The static analysis was conducted utilizing MobSF for several compelling reasons. Firstly, the framework is widely used within the security testing community [44]. Additionally, it is well maintained, open-source, and has the capability of identifying various vulnerabilities and risks within applications [45]. It also features a user-friendly interface, simplifying the analysis process, and comprehensively presents findings, aiding in understanding and addressing identified issues effectively.

During the analysis, MobSF performs a series of tests to assess various aspects of the mobile application's security and privacy. These tests cover multiple aspects of risks and vulnerabilities, and this study will focus on the tests specified in Table 3.2.1.

Table 3.2.1: MobSF Static Analysis Tests

Test	Description
Application permissions	Evaluate if the requested permissions align with the app's intended functions, detecting potential overprivileged permissions.
Network Security	Analyze network behavior for vulnerabilities like insecure protocols or unencrypted data transmission.
Certificate analysis	Checks digital certificates for issues like expiry or trust, highlighting security weaknesses.
Manifest analysis	Review the app's manifest for misconfigurations or vulnerabilities.
Code analysis	Conducts static analysis to identify security vulnerabilities or coding errors.
Abused Permissions	Flag permissions that are commonly abused by known malware.
OFAC-sanctioned countries	Check if the app communicates with sanctioned countries, indicating legal risks.
Trackers	Identifies third party trackers collecting user data without consent.
Hardcoded Secrets	Search for sensitive information directly embedded in the code, exposing security risks.

By uploading the APK file of an app to MobSF the static analysis is initiated and MobSF will proceed to test the application for security vulnerabilities, privacy concerns, and other potential issues that could compromise its integrity or users' data. Once the analysis is complete, MobSF will generate a comprehensive report of the findings and classify the application into different risk zones. MobSF initially assigns every app an ideal score of 100. However, this score changes based on the severity of each finding. A high-severity finding reduces the score by 15 points, a warning reduces it by 10 points, and a good finding increases it by 5 points. If the resulting score is greater than 100,

it is kept as 100. If it falls below 0, it is set to 10. Based on the final score, the App Security Score is classified into four risk levels: Critical Risk (0 - 15), High Risk (16 - 40), Medium Risk (41 - 70), and Low Risk (71 - 100).

MobSF is also prone to produce a significant amount of false positives for certain vulnerabilities [25, 44] as it may not always have access to the complete context of an application's behavior or environment. Therefore, the results obtained from the static analysis output by MobSF were verified manually by:

- **Examining the Code:**
  - Review the code of the applications to rule out false positives presented in the MobSF code analysis related to insecure number generators, ciphers, and cipher modes.
- **Evaluating Permissions:**
  - Assess whether permissions labeled as "dangerous" are essential for the app's intended functionality. This involves examining the protection level associated with each permission, as defined by Android's documentation[4].
- **Assessing Hardcoded Secrets:**
  - Identify and evaluate hardcoded sensitive information embedded within the application's source code, such as API keys, passwords, or cryptographic keys. Determine potential security risks and vulnerabilities associated with exposing such information.
- **Assessing Manifest Misconfigurations:**
  - Evaluate the severity of misconfigurations in the Android manifest such as allowing an application to be installed on vulnerable unpatched Android versions.
- **Evaluating Domains:**
  - Investigate the relevance of communication with domains originating from countries sanctioned by Office of Foreign Assets Control (OFAC). Assess the necessity and potential legal implications of such communication.

Manual Code Analysis

The manual code analysis was conducted through a systematic process to verify potential security and privacy vulnerabilities flagged by MobSF within the examined applications. This process involved the examination of specific code segments flagged by MobSF, such as insecure number generators, weak encryption, and other findings related to security and privacy.

[4]Android's documentation for permissions (https://developer.android.com/reference/android/Manifest.permission)

Each flagged file and associated code segments were reviewed to assess their relevance to security and privacy concerns. This evaluation entailed a thorough examination of the context in which the flagged code was implemented, considering its potential impact on the overall security posture of the application.

Analyzing the flagged code in each file aimed to identify and understand the underlying security implications, such as susceptibility to cryptographic attacks, exposure of sensitive data, or potential surfaces for exploitation by malicious actors.

Manual Permissions Evaluation

The evaluation of the app permissions involved two simple steps. First, permissions documentation was read to understand when and why a specific permission is needed. Secondly, the identified permissions were assessed to determine their relevance considering whether they were essential for the intended use and functionality of the apps.

Manual Hardcoded Secrets Assessment

The assessment of hardcoded secrets involved reviewing the list of provided secrets from MobSF, filtering for keywords like ”key,” ”secret,” ”token,” ”id,” ”oauth,” and ”API,” followed by an evaluation of the risks associated with the identified secrets.

Manual Manifest Evaluation

Evaluate the high-risk flags made by MobSF for relevance and security degree.

Manual OFAC Domain Evaluation

Evaluate if the communication is justified by being a known and secure domain or if it poses a potential risk of violating sanctions enforced by the OFAC.

3.2.2 Dynamic Security Analysis

Using the APK uploaded to MobSF during the static analysis, the dynamic analysis is conducted to further assess the application’s behavior and identify runtime-related security risks. Detailed information about the emulation setup can be found in subsection ”emulation” below. The dynamic analysis in MobSF captures runtime information such as network traffic, system calls, log messages, and API calls. These are then analyzed for potential security vulnerabilities, privacy issues, and other runtime-related risks. The output of the dynamic analysis includes detailed reports and logs that provide insights into the application’s runtime behavior, allowing for the identification and mitigation of security and privacy threats. The logs created are:

1. **Logcat Logs:** Logcat logs captured during dynamic analysis provide insights into the application’s behavior, including runtime information, error messages,

warnings, and debugging information generated by the Android operating system and the application itself.

2. **Dumpsys Logs:** Dumpsys logs collected during dynamic analysis contain information about the application's memory usage, process states, Central Processing Unit (CPU) utilization, battery usage, and other system-level metrics. These logs help identify performance issues and resource bottlenecks.
3. **HTTP(s) Logs:** HTTP(s) logs captured during dynamic analysis include network traffic generated by the application, such as HTTP requests and responses exchanged with remote servers. These logs provide insights into the application's communication patterns and interactions with external services.
4. **Application Data:** During dynamic analysis, various types of application data are collected, such as files accessed, databases queried, and inter-component communication events. Analyzing this data helps identify security risks related to data storage and insecure data handling.

Emulation

In the study, a Samsung Galaxy S10 device was emulated for the dynamic analysis. The device was running Android version 11.0 with API level 30 and had 8GB of RAM. It should be noted that MobSF is compatible with Android versions up to 11 and API level 30.

The Open GApps project was used to ensure smooth functionality and compatibility. Open GApps[5] provided necessary Google services and applications within the emulated environments.

The emulations were conducted on a Windows 11 machine equipped with an AMD Ryzen 7 7700X processor and 32GB of RAM using Genymotion [6].

Runtime Analysis/Data Collection

Each app was thoroughly examined for an estimated 15 minutes, giving plenty of time to go through all the app's activities and available features. During the 15-minute examination, all accessible activities were tested. The initial step involved using a fake email account to create a user account in the app and then using the app as intended by uploading images, scheduling workouts, editing entered information, and exploring all available features. Additionally, MobSF was utilized to identify all executable activities within the app, which were then run automatically. This comprehensive approach ensured a thorough assessment of each app's functionality and user experience.

It is worth noting that the apps were used only for their intended purposes, e.g., creating profiles, adding goals, scheduling workouts, changing user info, and so on. As

[5]Open GApps (https://opengapps.org/)
[6]Genymotion (https://www.genymotion.com/)

mentioned, we did not perform any interactions that could tamper with the app and its server-side infrastructure, e.g., interfere with or inject malicious inputs or network traffic. The user information used in the study is documented in table 3.2.2. Note that features that required a paid plan or similar were excluded from the study; only the free activities and interfaces were examined.

Table 3.2.2: Generic user information generated by ChatGPT

Attribute	Value
Name	John Doe
Username	johndoe3744
Email	johndoe3744@gmail.com
Password	ThisIsMyPassword123-
Date of Birth	January 1, 1990
Address	123 Main Street, Anytown, USA Florida
Phone Number	+1 (555) 123-4567

Domain Analysis

The domains found during the dynamic analysis are evaluated using the methods in section 3.2.3.

Log Analysis

The logs acquired from the MobSF dynamic analysis were examined manually to identify potential security vulnerabilities, abnormal behaviors, and privacy-related concerns within the analyzed application. However, Dumpsys logs were excluded from this study since they primarily provide system information such as activity states, service statuses, and memory usage.

Firstly, the HTTP(S) logs were examined to determine if the app's traffic was transmitted securely and to investigate any insecure channels that might expose sensitive user information, such as that presented in Table 2.0.1. To identify any disclosed user information, the HTTP(S) logs were examined by using various keywords, which included: "@gmail.com", "api_key", "username", "password", "address:", "login", "post" as well as the user information used to create the account (see Table 3.2.2).

Similarly, the Logcat logs were also examined for any insecure log entries with the same keywords as the HTTP(S) logs. The Logcat logs were also examined to identify whether they reveal any sensitive information about the user's app usage patterns, activities, or web traffic. These logs may also expose any tokens, credentials, or other sensitive details the app utilises. Therefore, additional keywords were used, such as: "select", "click", "touch", and "input". It is important to note that the Logcat logs generated on the device are accessible to other apps running concurrently, which could potentially lead to the exposure of sensitive information to those apps [27].

Finally, the app data were analyzed using SQLite Browser, and each stored database query was manually checked for privacy risks and user information.

3.2.3 Server-Side Analysis

Previous studies [57] have emphasized the importance of analyzing an application's communication channels due to the potential for data leakage and unauthorized access to sensitive user information. By examining the security of a domain's communication configuration, this study aims to identify any instances of transmission that may compromise user privacy or security.

Therefore, a web server analysis was performed on each domain discovered during the dynamic analysis using the free online tool Qualys SSL Labs[7] to assess the security levels of the HTTPS data transmissions. Qualys SSL Labs is a web-based service that offers tools and assessments for analyzing web servers' SSL/Transport Layer Security (TLS) configuration. Through the tests, the tool can identify vulnerabilities and weaknesses in organizations' SSL/TLS implementation, ensuring the security and privacy of their web applications and services. The analysis provides an overall rating of the web server's security (A+, A, B, C, D, E, F, T) and a score and potential weaknesses for its certificate, protocol support, key exchange, and cipher strength. It is important to note that only the root domains were tested, and subdomains were not individually assessed.

3.2.4 Privacy Policies Analysis

Privacy policies are an important part of applications, especially where users entrust their personal information to developers and service providers. These policies act as a foundational agreement between users and organizations, describing how personal data is collected, utilized, stored, and shared [31].

In order for users to make informed decisions about whether or not to engage with an application or service, the privacy policies must be comprehensive and easily accessible. Previous studies have revealed that many apps have privacy policies that are difficult to understand and require higher levels of education to comprehend [25, 58]. In many cases, it was also found that a user had to perform multiple actions even to find the privacy policies [54]. Furthermore, according to the findings of [58], many apps show potential non-compliance with both their stated privacy policies and certain GDPR guidelines. This observation is further supported by the findings presented in [31].

Based on existing research, this study adopts a privacy policy analysis that assesses the accessibility, GDPR compliance, readability, and also possible unfair clauses of the privacy policies within the selected apps.

[7] Qualys SSL Labs (https://www.ssllabs.com/ssltest/index.html)

Unfair Clauses, Availability & Readability

To assess potential unfair clauses within the privacy policies of the selected apps, CLAUDETTE[8], a web-based tool, was utilized. Developed for research purposes, CLAUDETTE categorizes potential unfair clauses into different categories based on privacy policies submitted as input. Additionally, a readability test was conducted using the WebFX[9] Readability Tool to evaluate the clarity and comprehensibility of the policies. Furthermore, a manual evaluation determined the availability of the privacy policies, examining whether they were accessible via Google Play or within the respective applications themselves.

GDPR Compliance & Policy Evaluation

To assess compliance with GDPR regulations, the online tool GDPRWise[10] was utilized. This tool automatically examines privacy policies for potential violations of GDPR, providing details on any identified non-compliance issues along with severity ratings.

Additionally, a manual evaluation was conducted to compare the privacy policies and information available on the Google Play Store with the data obtained from the dynamic analysis.

3.3 Responsible Disclosure Process

In this study, we follow similar responsible disclosure procedures as proposed in [25]. After completing the vulnerability assessment, individual reports outlining the findings for each application were compiled. These reports were then emailed to the respective companies and/or developers, utilizing the contact information available on the Google Play Store platform or websites. The identified issues were communicated to the companies/developers 60 days before the study was made public, allowing time for them to mitigate the vulnerabilities. Notwithstanding, all the results from this research are published in a de-identified manner, without disclosing the names of the apps or companies.

[8]CLAUDETTE (http://claudette.eui.eu/demo/index.html)
[9]WebFX Readability tool (https://www.webfx.com/tools/read-able/)
[10]GDPRWise (https://gdprwise.eu/policy-checker/)

Chapter 4

Results

This chapter presents a detailed collection of the security and privacy-related risks found during the static, dynamic, server-side, and privacy policy analyses (see Chapter 3). The findings range from vulnerabilities in communication protocols to inconsistency in privacy policy adherence. By exposing each aspect of these findings, these results aim to offer valuable insights into the security and privacy posture of mHealth applications.

4.1 App Selection

Ten apps met all the required criteria from the initial set of the top 30 apps across four English-speaking countries. The selected top-ranked apps belong to the health and fitness category and originate from four countries, representing various demographics and user preferences within the health and fitness sector.

Table 4.1.1 presents a breakdown of the download distribution, showing the number of apps falling under the variety of different download ranges. Cumulatively, the selected apps have been downloaded over 237 million times, underscoring the relevance of studying them specifically, even if they constitute a small sample of ten apps.

This study aims to evaluate privacy and security risks associated with health and fitness apps and make recommendations on minimizing them. However, it is important to note that any information that could identify the apps being evaluated will not be shared. The study aims to assess and improve app security overall rather than exposing individual developers or apps.

4.2 Static Security Analysis

The static analysis was conducted using the MobSF to identify potential security vulnerabilities and weaknesses within the codebase of the tested mHealth apps. This section presents the key findings from the static analysis, including assessing

Table 4.1.1: Number of Apps by Downloads.

Number of Apps	Downloads
1	100,000,000+
2	50,000,000+
2	10,000,000+
3	5,000,000+
2	1,000,000+
10	237,000,000+

hardcoded secrets, insecure code patterns, misconfiguration, and permission evaluation.

4.2.1 Hardcoded Secrets

Hardcoding sensitive information such as API keys or cryptographic materials directly into mobile application source code could pose significant security risks. These hardcoded secrets can easily be extracted through reverse engineering, enabling attackers to gain unauthorized access to backend systems or compromise user data. Moreover, mobile applications are susceptible to binary attacks, where attackers analyze the application's binary code to uncover embedded secrets, making hardcoded secrets an increased security risk [39, 40].

The static analysis revealed several instances of hardcoded secrets within the code of the tested mHealth apps. While some of these secrets may not pose an immediate security threat, their presence highlights poor coding practices and potential vulnerabilities. Table 4.2.1 summarizes the findings related to hardcoded secrets.

Table 4.2.1: Summary of Hardcoded Secrets.

Category	Findings	Risk
API Keys	Every app in the study was found to have hardcoded API keys.	Hardcoding API keys increase the risk of exposure and unauthorized access to sensitive data or functionalities.
Database Uniform Resource Locator (URL)s	Eight hardcoded database connection URLs were identified within the source code.	Carries no immediate risk by itself.
Client Secrets & Tokens	Six instances of hardcoded client secrets and tokens used for authentication with providers such as Facebook and other third parties were found.	Hardcoding authentication credentials increase the risk of account takeover and unauthorized access to user data.

Storing API keys, database credentials, identity, and access management permissions directly in the source code increases the risk of exposing sensitive data or functionalities to unauthorized access [39]. Nonetheless, this study has identified multiple instances of each of these issues within some of the most widely used health and fitness apps from the USA, United Kingdom, Australia, and Canada.

While these hardcoded secrets might not pose an immediate security threat on their own [39], such as a database URL, their collective prevalence is worth noting. In this study, it was found that eight out of ten apps had hardcoded their Firebase URL (8/10 apps) as well as the corresponding API key (10/10 apps). Although Firebase API keys are not used for authorization and may not directly lead to unauthorized access [11], their widespread use underscores the importance of secure coding practices to minimize potential risks and ensure the overall security of the applications.

The prevalence of different types of keys in the apps included in this study is shown below, along with a brief description of each.

1. **Third Party API Keys:**
    - Across the tested apps, many third party API keys were found hardcoded within the source code. While this practice is generally discouraged due to the risk of exposure, the impact varies depending on the sensitivity of the API and the permissions granted.
2. **Google API Keys:**
    - Notably, each app had its Google API key and its Google Crash Report key hardcoded within the source code. While it is generally recommended to avoid this practice to minimize the risk of unauthorized access and misuse, it is important to recognize that the Google API keys found in the tested apps are primarily intended for service access rather than authorization. However, safeguarding these keys remains crucial to prevent potential misuse that could lead to spamming and billing linked to the API key or the disclosure of location information, thereby posing a privacy risk to users.

API keys are crucial in controlling access to services and functionalities within applications. Compromised API keys can lead to unauthorized access to sensitive data or functionalities within an application [2]. Attackers with access to API keys may bypass access controls and gain unrestricted access to protected resources. In addition, compromised API keys can also be leveraged in Distributed Denial of Service (DDoS) attacks, rendering it inaccessible to legitimate users [2].

Unprotected stored variables are recognized as common security risks and are listed in OWASP's Mobile Top 10 [39]. Mitigation tactics for proper secrets management are easily available from multiple sources, including Google and OWASP [12, 43], which offers best practices, guides, and instructions for implementing proper secrets management and securely using API keys. Giving developers no reason to leave secrets unprotected and hardcoded within the source code or littered throughout

configuration files.

Synthetic finding:

- All ten apps disclosed some hardcoded API key in their code, while eight apps also disclosed their database URL and six apps other client secrets and tokens.

4.2.2 Code Analysis

Examining mobile application code for vulnerabilities is essential for upholding the security and reliability of mHealth apps. Malicious actors can easily access and browse application binaries with tools such as MobSF, simplifying the reverse engineering process. Hence, vulnerabilities such as weak random number generators or insecure encryption configurations become easy targets for exploitation [25, 39]. Weak random number generators and insecure encryption configurations, including the utilization of outdated or vulnerable algorithms such as ECB mode, may introduce predictability into the app's cryptography, leaving the application susceptible to Man in the Middle (MitM) attacks, which may lead to session hijacking, data interception, data manipulation, and unauthorized access[37]. The findings from the investigated high-risk flags supplied by MobSF during the static analysis are summarized in Table 4.2.2.

Table 4.2.2: Insecure Code in Apps according to MobSF static analysis.

App	High Risk Issues
App1	App uses the encryption mode Cipher Block Chaining (CBC) with PKCS5/PKCS7 padding. This config is vulnerable to padding oracle attacks.
App2	1. Debug config enabled. Production builds must not be debuggable. 2. Remote Web debugging is enabled.
App3	App uses the encryption mode CBC with PKCS5/PKCS7 padding. This config is vulnerable to padding oracle attacks.
App4	Uses ECB mode for encryption.
App5	Debug config enabled. Production builds must not be debuggable.
App6	-
App7	1. App uses the encryption mode CBC with PKCS5/PKCS7 padding. This config is vulnerable to padding oracle attacks. 2. Remote Web debugging is enabled.
App8	CBC with PKCS5/PKCS7 padding, ECB mode
App9	-
App10	App uses the encryption mode CBC with PKCS5/PKCS7 padding. This config is vulnerable to padding oracle attacks.

The analysis revealed several instances of insecure encryption configurations, posing significant and urgent security risks. Five applications were found to utilize the encryption mode CBC with PKCS5/PKCS7 padding, which is known to be vulnerable to padding oracle attacks under certain conditions. This is best described in the OWASP Web Security Testing Guide as follows:

"The padding oracle attack enables an attacker to decrypt encrypted data

> *without knowledge of the encryption key and used cipher by sending skillfully manipulated ciphertexts to the padding oracle and observing the results returned by it. This causes loss of confidentiality of the encrypted data. E.g. in the case of session data stored on the client-side the attacker can gain information about the internal state and structure of the application.*
> *A padding oracle attack also enables an attacker to encrypt arbitrary plain texts without knowledge of the used key and cipher. If the application assumes that integrity and authenticity of the decrypted data is given, an attacker could be able to manipulate internal session state and possibly gain higher privileges."* [38]

While using CBC with PKCS5/PKCS7 padding indicates a potential problem, it is worth noting that this is only an indication and may result in false positives. MobSF, the tool used in this analysis, is known to provide false positives for vulnerabilities, including those related to encryption configurations. A deeper analysis is necessary to determine the actual risk.

To perform a padding oracle attack, additional conditions must be met. The application must return distinguishable error messages or responses based on the validity of the padding. If the ciphertext is authenticated with a Message Authentication Code (MAC), the risk of a padding oracle attack can be significantly reduced.

Therefore, while the detection of CBC with PKCS5/PKCS7 padding highlights a potential security issue, it should be interpreted with caution. Further investigation is required to confirm the vulnerability and assess the overall impact on the application's security.

Additionally, one application was found to use ECB mode for encryption (see figure 4.2.1), further increasing security concerns. ECB divides the plaintext into blocks and encrypts each block independently, and due to its deterministic nature, where identical plaintext blocks always result in identical ciphertext blocks [30], ECB mode encryptions are vulnerable to pattern recognition and chosen plaintext attacks. Due to its weak encryption it is not recommended for developers to implement AES/ECB with or without padding [13, 41] but instead to use AES/Galois/Counter Mode (GCM)/NoPadding as recommended by Google [13]. However, it is unclear from the code analysis what information is transmitted using the insecure encryption modes. Nevertheless, the prevalence of insecure encryption configurations across the analyzed applications is alarming and requires immediate attention.

Moreover, it was discovered that three of the applications had debug configurations enabled, allowing for remote web debugging and production build debugging [40]. Enabling debug configurations in production builds could expose sensitive information and functionality to potential attackers. Debuggable production builds simplify the task for malicious actors to intercept and manipulate application behavior, potentially resulting in unauthorized access to sensitive data.

```
public static byte[] h(byte[] bArr, String str) {
  try {
      SecretKeySpec b10 = b(str);
      System.out.prinln(b10);
      Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
      cipher.init(1, b10);
      return cipher.doFinal(bArr);
  } catch (Exception e10){
      e10.printStackTrace();
      retrun null;
  }

}
```

Figure 4.2.1: Use of ECB mode by App4.

Notably, the code analysis also found several false positives regarding random number generators, MD5 and SHA1 usage. These findings were discarded as weak cryptographic strategies were used in non-cryptographic use cases (e.g., hashes are used for integrity checks of downloads).

Synthetic findings:

- Insecure encryption configurations, including CBC mode with PKCS5/PKCS7 padding (five apps) and ECB mode (two apps), were identified in several applications.
- Debug configurations enabled in production builds were found in three applications, potentially exposing sensitive information and functionality to attackers.

4.2.3 Manifest

The manifest file of Android applications contains critical information about the application, including its permissions, components, and requirements. Analyzing the manifest file helps identify potential security issues and vulnerabilities that could compromise the integrity and security of the application. Table 4.2.3 summarizes the findings from the manifest analysis.

The analysis of the manifest revealed two prominent issues. Firstly, four applications were found to have the App link asset verification URL missing or configured incorrectly. App Links allow redirection from a web URL or email to the mobile app. If the asset verification URL is missing or misconfigured, it opens up the possibility of a malicious app hijacking such URLs, potentially leading to unauthorized access or exploitation [3].

Secondly, seven applications were identified as being installable on older Android versions. This practice is concerning as older Android versions often have multiple unfixed vulnerabilities and may not receive regular security updates.

Table 4.2.3: Manifest issues in Apps.

N. of apps	Manifest Issues	Description
4	App link asset verification URL not found or configured incorrectly	App Links allow users to redirect from a web URL/email to the mobile app. A malicious app can hijack such URLs if this file is missing or incorrectly configured for the App Link host/domain.
7	Can be Installed on Older Android versions	Older Android versions that have multiple unfixed vulnerabilities and will not receive reasonable security updates from Google.

Synthetic findings:

- Four applications were found to have missing or incorrectly configured app link asset verification URLs, posing a risk of malicious app hijacking and unauthorized access.
- Seven applications were identified as installable on older Android versions, which is concerning due to unfixed vulnerabilities and lack of regular security updates on older Android versions.

4.2.4 Permissions Evaluation

Permissions are crucial in defining the level of access an Android application has to various device resources and functionalities. Android classifies certain permissions as "dangerous," indicating that their misuse or overprivilege could potentially compromise user privacy or device security [9]. Overprivileged apps pose risks in terms of both user privacy and security. Unnecessary access to sensitive device features (e.g., external storage, photos and media, contacts list) increases the attack surface for potential exploitation by malicious actors, potentially leading to a breach of user privacy [46]. The most common permissions used by the apps are shown in Table 4.2.4.

Most of the permissions found to be used by the apps could be justified and aligned with the apps' intended purposes. However, certain permissions such as `BLUETOOTH_SCAN`, `WRITE_CONTACTS`, and `GET_TASKS` indicate overprivilege and raise concerns regarding potential misuse.

The `GET_TASKS` permission, although deprecated in API level 21 and no longer enforced, allowed applications to retrieve information about currently and recently running tasks. This permission could potentially enable malicious applications to discover private information about other applications [6].

Similarly, the `WRITE_CONTACTS` permission allows an application to modify the contact data stored on a user's phone. It was also found that the permission was unimportant for the apps' intended use and should be removed immediately.

Furthermore, the BLUETOOTH_SCAN permission is required for an application to discover and pair nearby Bluetooth devices. While this permission might be needed to pair different activity trackers, the app had no such functionality. However, the app in question is downloadable on different devices, and this permission could be needed for the app to function on their smartwatch app. Nevertheless, the permission seems unnecessary for the mobile app alone.

However, some permissions collectively raised concerns, suggesting potential overprivilege even if their individual prevalence could be justified. For instance, four of the apps requested for both READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE while also requesting READ_MEDIA_IMAGES (4/10 apps), READ_MEDIA_AUDIO (4/10 apps), and READ_MEDIA_VIDEO (3/10 apps) permissions, which should suffice for the app's media-related functionalities. This redundancy suggests a lack of optimal permission management, potentially exposing user data to unnecessary risks.

Furthermore, the evaluation identified some location-related permissions across the examined mHealth applications. Notably, five apps were found to request both ACCESS_COARSE_LOCATION (4/10 apps) and ACCESS_FINE_LOCATION (5/10 apps) permissions, along with additional permissions such as ACCESS_MEDIA_LOCATION (3/10 apps). While it is reasonable for fitness-oriented apps to utilize location tracking in the context of, for example, running, the need for multiple location tracking permissions seems excessive and unnecessary. It is also worth noting that the ACCESS_MEDIA_LOCATION permission was only justifiable in one of the three cases.

Synthetic findings:

- Four apps had multiple storage-related permissions, where only a few would suffice for the apps' intended use. Suggests a lack of optimal permission management.
- Four apps had multiple location-related permissions, where including multiple permissions was unnecessary for most cases.

4.2.5 Trackers

The examined applications were found to communicate with various trackers, as summarized in Table 4.2.5. These trackers serve various purposes: analytics, advertisement, crash reporting, and identification. Trackers such as Braze (formerly Appboy), AppsFlyer, and Optimizely were among the most frequently encountered.

The evaluation of trackers used by the apps revealed several notable findings. Firstly, it was found that at least half of the apps utilize cross-device advertisement networks. Sharing information with trackers can raise privacy and security concerns, particularly those used for user profiling and advertisement. By leveraging user data from browsing states, cookies, and browsing history, cross-device ad networks can deliver targeted ads across various devices [48].

Table 4.2.4: Most common dangerous permissions used by apps.

N. of Apps	Dangerous Permission
10	android.permission.POST_NOTIFICATIONS
8	android.permission.READ_EXTERNAL_STORAGE
8	android.permission.WRITE_EXTERNAL_STORAGE
5	android.permission.ACCESS_FINE_LOCATION
4	android.permission.CAMERA
4	android.permission.READ_CONTACTS
4	android.permission.ACCESS_COARSE_LOCATION
4	android.permission.READ_MEDIA_AUDIO
4	android.permission.READ_MEDIA_IMAGES
4	android.permission.READ_CALENDAR
4	android.permission.USE_CREDENTIALS
3	android.permission.ACTIVITY_RECOGNITION
3	android.permission.READ_MEDIA_VIDEO
3	android.permission.ACCESS_MEDIA_LOCATION
2	android.permission.RECORD_AUDIO
2	android.permission.BLUETOOTH_CONNECT
2	android.permission.BODY_SENSORS
1	android.permission.GET_TASKS
1	android.permission.GET_ACCOUNTS
1	android.permission.AUTHENTICATE_ACCOUNTS
1	android.permission.READ_MEDIA_VISUAL_USER_SELECTED
1	android.permission.WRITE_CALENDAR
1	android.permission.BLUETOOTH_SCAN
1	android.permission.WRITE_CONTACTS

Furthermore, the analysis identified several apps engaging in user profiling activities using specific trackers such as Tealium, Braze, and Amplitude. These trackers are known for their capabilities in analyzing user behavior and preferences to create detailed user profiles, potentially raising concerns about user privacy and data protection.

Notably, one tracker, Mixpanel, was found to have a concerning privacy breach clause in its policy. According to an analysis by Exodus Privacy [1], Mixpanel's policy states that end-users tracked by MixPanel's "customers" have no right to delete their personal information[2]. This indicates potential privacy implications for users, such as negating data deletion rights, intervenability, and objection to processing.

Additionally, certain trackers such as Branch [3] and Inmobi [4] are known to collect device-specific information according to Exodus Privacy, including unique identifiers and other device-related data. This raises concerns about the extent of data collection by these trackers and the potential for user privacy infringement.

[1]Exodus Privacy (https://reports.exodus-privacy.eu.org/en/trackers/)
[2]Exodus Privacy (https://reports.exodus-privacy.eu.org/en/trackers/118/)
[3]Exodus Report on Branch (https://reports.exodus-privacy.eu.org/en/trackers/167/)
[4]Exodus Report on Inmobi (https://reports.exodus-privacy.eu.org/trackers/106)

Lastly, most apps communicate with well-known analytics trackers, including Google Crashlytics, Google Firebase Analytics, and Facebook Analytics. These trackers are commonly employed to collect and analyze app usage data, providing developers with insights into user behavior and app performance.

Synthetic findings:

- At least half of the apps utilize cross-platform advertisement networks, raising privacy and security concerns regarding user profiling and targeted advertising.
- Most apps communicate with well-known analytics trackers, such as Google Crashlytics, Google Firebase Analytics, and Facebook Analytics, for collecting and analyzing app usage data to provide insights into user behavior and app performance.

Table 4.2.5: Number of Apps by Tracker.

N. of Apps	Tracker	Used For
7	Google Firebase Analytics	Analytics
7	Google CrashLytics	Crash reporting
7	Facebook Analytics	Analytics
6	Facebook Login	Identification
5	Facebook Share	Sharing
4	Braze (formerly Appboy)	Analytics, Advertisement, Location
3	Branch	Analytics
3	Google AdMob	Advertisement
3	AppsFlyer	Analytics
3	Optimizely	Analytics
3	New Relic	Analytics
2	IAB Open Measurement	Identification, Advertisement
2	Sentry	Crash reporting
1	MixPanel	Advertisement, Analytics
1	Bugsnag	Crash reporting
1	Amplitude	Analytics, Profiling
1	AccountKit	Analytics
1	Tealium	Analytics
1	Pendo	Analytics
1	OpenTelemetry	Analytics
1	Singular	Analytics
1	Urbanairship	Analytics
1	Demdex	Analytics
1	Inmobi	Advertisment

4.3 Dynamic Security Analysis

The dynamic analysis involved utilizing various tools and techniques to evaluate the runtime behavior of the mHealth apps under examination. By monitoring

Figure 4.3.1: Email used as an identifier in a RESTful URI.

app activities during execution, potential security threats and vulnerabilities were identified. This section presents the results from the dynamic analysis findings, which bring insights into network communication, data transmission, and interaction with external entities.

4.3.1 HTTP(S) Traffic Analysis

The HTTP(S) traffic analysis yielded reassuring results as it was found that all apps transmitted sensitive user information over encrypted HTTPS connections, protecting the security of users' data against eavesdropping and unauthorized access.

However, despite employing a secure communication protocol, two apps were found to use email addresses as identifiers in RESTful URI requests, as depicted in Figure 4.3.1. Additionally, a third app similarly revealed users' usernames through this method. This practice is generally considered insecure, as using email addresses or usernames in RESTful URI requests exposes user information directly in the request path. Although the contents of HyperText Transfer Protocol Secure (HTTPS) requests, including paths and query parameters, are encrypted and not exposed during transmission, this information may still be logged on the server side. Server logs could, therefore, contain sensitive user information, which poses a risk to user privacy if the logs are accessed by unauthorized individuals or if proper log management practices are not followed.

Moreover, the analysis revealed that four apps transmitted users' passwords in cleartext POST requests. This finding points to the apps possibly storing passwords in plaintext, which may result in a system compromise [42].

Synthetic findings:

- All apps were found to use secure and encrypted communication.
- Two apps were found to disclose user email addresses in GET requests, and a third app revealed users' usernames through similar means.
- Four apps were found to transmit users' passwords in cleartext POST requests, indicating a potential security risk.

4.3.2 Logcat & Appdata Analysis

Developers primarily use the Android Logcat logs to debug applications during development. Logcat logs provide insights into the runtime behavior of an application, aiding developers in identifying and resolving errors, crashes, or unexpected behaviors [33]. However, developers must remove any debugging prints before deploying the application to production to prevent accidentally logging user actions or network requests.

During the analysis of the Logcat logs generated in the dynamic analysis, it was discovered that three applications logged user actions (see Figure 4.3.2). Logging user actions raises privacy concerns, as it may capture user interactions without explicit consent, and all other apps running on the device have full access to such logs. Furthermore, one application was found to log POST values, including information such as country, device details (including OS version), local IP, and session_id for the tracker Branch (see Figure 4.3.3), which might make the connection vulnerable to session hijacking [4]. This poses privacy risks by exposing user-related data and potential security vulnerabilities by disclosing sensitive information to other apps on the phone. The app data analysis yielded no significant results.

Synthetic findings:

- Three applications were found to log user actions in the Logcat logs.
- One app was found to log POST values, including information such as country, device details, local IP, and session id for the tracker Branch.

4.4 Server-Side Analysis

During the dynamic analysis, it was observed that some apps communicated with several domains within the short duration of their use. Table 4.4.1 presents the count of domains communicated with by each app during the dynamic analysis.

It is apparent from the table that there is considerable variation among the apps in terms of the number of domains they interact with. For instance, App3 and App9 communicated with relatively fewer domains (0 and 15, respectively), while App8 and App10 were found to interact with a significantly higher number of domains (236 and 240, respectively). Altogether, the ten apps communicated with 404 unique domains, excluding subdomains.

The large diversity in domain communication patterns is concerning, as there seems to be no justifiable reason for an application to communicate with such a large number of domains within a short timeframe. This behavior raises potential privacy and security concerns, as apps interacting extensively with domains may accidentally increase the risk of data leakage. Further examination of these communications and the related domains is essential to measure the extent of the risks associated with such

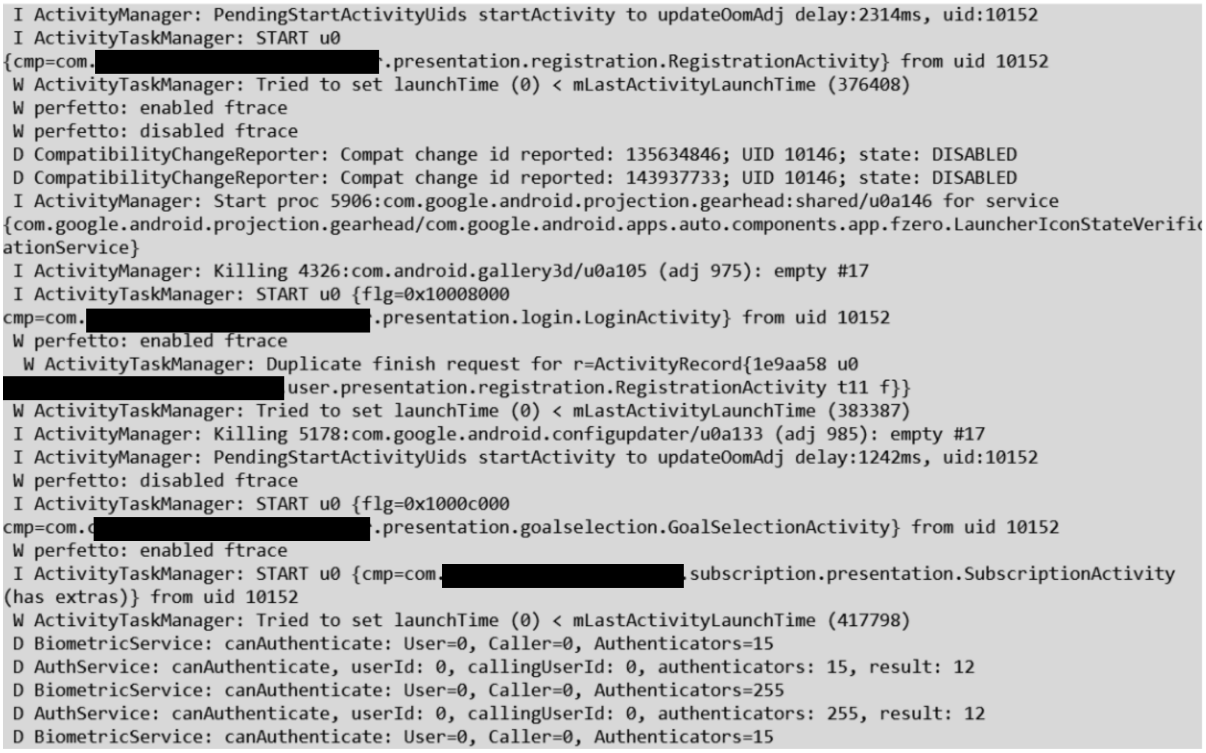

Figure 4.3.2: App1 Logcat User actions logged. As seen in the image, it is possible to outline when an activity is started, finished, or killed (i.e., see `ActivityManager` and `ActivityTaskManager`).

Figure 4.3.3: POST values and branch session_id logged in Logcat logs. The image shows a screenshot of the Logcat logs which discloses information that should not be logged.

Table 4.4.1: Domains communicated with during dynamic analysis.

Apps	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
Domains	17	30	0	111	34	18	33	236	15	240

behavior.

To ensure the security of the domains communicated with by the apps, their TLS/SSL configurations were assessed using Qualys SSL. The summarized results are presented in Figure 4.4.1. It is important to note that any domain scoring a B or below should evaluate its configurations to enhance security measures, as there is no reason to keep servers configured with lower security levels. Of particular concern is the prevalence of 2.5% (10 domains) receiving a score of T, indicating multiple misconfigurations or vulnerabilities. Considering the popularity of the tested apps, communication with domains with such vulnerabilities requires immediate attention to mitigate potential security risks and should be evaluated promptly.

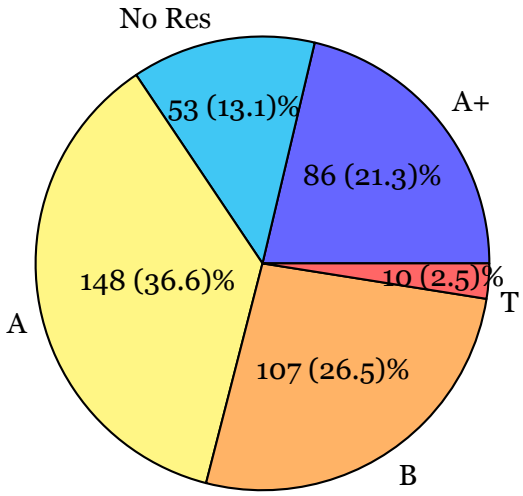

Figure 4.4.1: Summary of Qualys SSL results from web server analysis ($n = 404$).

During the static analysis, certain domains were found to be communicated with by the apps but not during runtime. This raises questions about the safety or potential risks associated with such communication. Among the domains identified (see Table 4.4.2), all originate from China, which is a OFAC-sanctioned country. However, communication with domains from OFAC-sanctioned countries may not necessarily pose an immediate risk. Further investigation is warranted to assess the nature and purpose of this communication.

It is worth mentioning that during the evaluation of the TLS/SSL configurations, `baidu.com` performed the worst, receiving a C grade from Qualys SSL. In contrast, `a.app.qq.com` achieved a B grade, while `wap.scjgj.sh.gov.cn` received the highest rating with an A. Note that these three domains were evaluated separately from the others and are not included in Figure 4.4.1.

Table 4.4.2: Communication between Apps OFAC Sanctioned Countries.

App	Domain	Country	Location
App 4	`www.baidu.com`	China	Hong Kong
App 6	`a.app.qq.com`	China	Beijing
	`wap.scjgj.sh.gov.cn`	China	Shanghai

Synthetic findings:

- Altogether, the ten apps communicated with 404 unique domains, excluding subdomains. Two apps communicated with over 230 domains each and a third with over 100 domains.
- 118 domains received a TLS/SSL configuration score of lower than A.
- Two apps communicated with a total of three domains from OFAC sanctioned countries.

4.5 Privacy Policies Analysis

Privacy policies play a vital role in mobile applications, informing users of the collection, processing, and security of their data. These policies must be fair, easily accessible, and understandable to users so they can make informed decisions about their data.

The results from the WebFX tests found that all apps scored between 30-50 in the readability index, meaning that their privacy policies are difficult to read, requiring college-level education. All the individual app results can be found in Table 4.5.1.

Table 4.5.1: WebFX Readability results

App	Result (x/100)	Readability age
App 1	31.8	21 - 22 y.o.
App 2	49	17 y.o.
App 3	41.2	19 - 20 y.o.
App 4	38.9	19 - 20 y.o.
App 5	40.1	18 - 19 y.o.
App 6	37.4	19 - 20 y.o.
App 7	43.2	18 - 19 y.o.
App 8	41.8	18 - 19 y.o.
App 9	41.2	18 - 19 y.o.
App 10	47.7	16 - 17 y.o.

The risks related to unfair or insufficient policies found using CLAUDETTE uncovered several unfair clauses in the evaluated apps' privacy policies. Figure 4.5.1 summarizes the results obtained using CLAUDETTE. Three apps had "unilateral termination" clauses, allowing the app provider to terminate the agreement without user consent

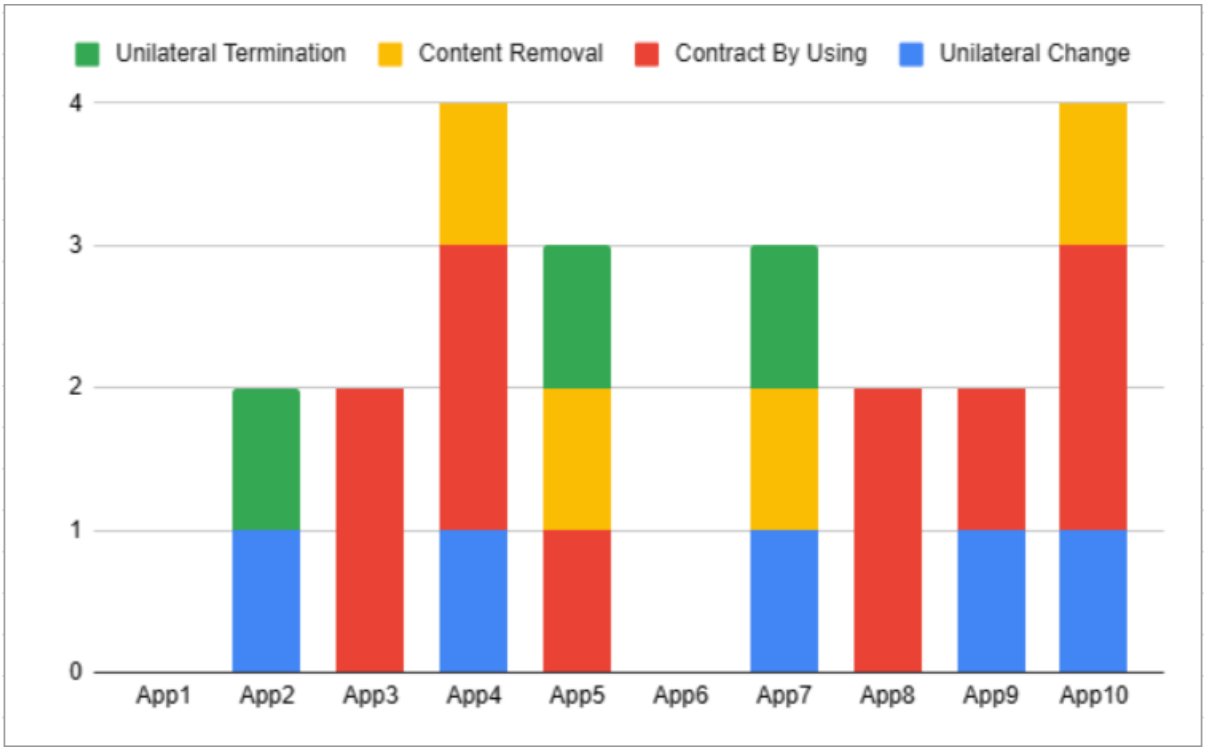

Figure 4.5.1: Summary of CLAUDETTE results of unfair clauses.

under certain conditions. Additionally, five apps were found to have "unilateral change" clauses, granting the app provider the authority to modify the privacy policy terms without user agreement. Furthermore, four apps have "content removal" clauses, giving the app provider the right to remove user-generated content without notification. Moreover, six apps featured "contract by using" clauses, meaning the user agrees to the privacy policy by merely using the app.

The manual availability analysis showed that all the apps had a functional privacy policy link accessible in their Google Play Store. However, the availability of the privacy policy within the app varied, with two apps lacking an accessible link on the login/signup page and six apps requiring multiple user interactions to reach them.

Moreover, information regarding how user data is collected and handled by the app, which can be found on the apps page in the Google Play Store, raised some concerns. All the app pages stated that: *"The developer has provided this information about how this app collects, shares, and handles your data"* and in nine out of ten cases, the page also stated that *"Data is encrypted in transit Your data is transferred over a secure connection"*. The results from section 4.3.1 confirm this statement. However, all app developers were transparent about the data their apps may collect.

Five of the ten apps claimed they do not share data with third parties. However, upon further investigation, it was found that two of those five apps communicated with over 230 different domains and one with over 100 domains. These findings cast doubt on the validity of the claim made by those three apps.

Lastly, the results regarding GDPR compliance, which was evaluated using GDPRWise, showed that all apps' privacy policies complied with GDPR.

Synthetic findings:

- Privacy policies of the examined apps scored between 30-50 in the readability index, indicating difficulty in comprehension without a college-level education.
- CLAUDETTE analysis revealed unfair clauses in privacy policies, including unilateral termination, unilateral change, content removal, and contract by using clauses across multiple apps.
- Manual availability analysis showed all apps had a functional privacy policy link in the Google Play Store, but accessibility within the app varied.

4.6 Responses to the Responsible Disclosure

Despite the proactive outreach, the response rate from the contacted stakeholders varied. Out of all the companies approached only App7 acknowledged the findings and responded. The response indicated that the identified vulnerabilities had been forwarded to the relevant teams for further action. The communication also emphasized the company's commitment to prioritizing privacy and security concerns. Remarkably, the remaining companies did not respond to the disclosure email, indicating a lack of engagement with the identified security issues.

Chapter 5

Discussion

The study's findings provide valuable insights into the current privacy and security posture of top-ranked health and fitness mHealth apps, allowing us to answer the research questions: *What are the security and privacy risks in top-ranked health and fitness apps? How can these risks be identified and mitigated through empirical studies?* This study is particularly significant as it represents one of the first attempts to review these top-ranked apps' security and privacy posture comprehensively. It offers a unique perspective on the prevalent risks and vulnerabilities they may pose to users. Each subsection, therefore, discusses each of these risks accordingly: (1) Misconfigurations & Permissions, (2) Apps' security testing & coding, (3) Insecure Communication, (4) Domains & Trackers, and (5) Privacy policies.

5.1 Misconfigurations & Permissions

During the static analysis, we found that most apps (8/10) had some manifest misconfiguration, either allowing the app to be installed on older Android versions (7/10 apps) or missing / incorrectly configured App link asset verification URLs (4/10 apps). However, to exploit the app link vulnerability requires specific circumstances[1], which might not make it the most attractive entry-point for an active attacker. On the other hand, allowing an app to be installed on older vulnerable versions of Android might open up another dimension for an attacker, allowing them to exploit known vulnerabilities in the systems, which might lead to unauthorized access to the app and its data or worse.

The static analysis also yielded valuable insight into the dangerous permissions used by the apps, showing relatively positive results given that only three apps were found to have one permission each, which did not reflect the app's intended use, resulting in an average of 0.33 unnecessary permissions per app ($n = 10$). Previous studies such as [25] discovered that, on average, 4.1 permissions of the examined apps' ($n = 27$)

[1]Link Hijacking (`https://www.cybersecurity.blog.aisec.fraunhofer.de/en/android-app-link-risks/`)

were unnecessary. Developers can adopt several strategies to mitigate the prevalence of permissions not required for the app's intended use. Firstly, it is recommended that app developers request permissions "in context," i.e. when the user starts to interact with the feature that requires it [25]. Additionally, if permissions are not essential for the app's functionality, they could be disabled by default, running the app with the highest level of privacy.

However, the prevalence of multiple storage permissions raised some concerns about their relevance to the apps' purposes. `READ_EXTERNAL_STORAGE` and `WRITE_EXTERNAL_STORAGE` were used by eight apps while some of them also requested `READ_MEDIA_IMAGES` (4/10 apps), `READ_MEDIA_AUDIO` (4/10 apps), and `READ_MEDIA_VIDEO` (3/10 apps). According to Android's permissions documentation[2] [34], starting in API level 33, the permission `READ_EXTERNAL_STORAGE` has no effect and developers are instead advised to use the other `READ_MEDIA` variations mentioned above. It would benefit developers to shift their usage of `READ_EXTERNAL_STORAGE` to the `READ_MEDIA` variations to align with best practices and ensure compatibility with future Android versions.

Furthermore, three apps were found to request multiple location permissions, i.e., `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION`, and `ACCESS_MEDIA_LOCATION`. These permissions grant apps access to various levels of location information, such as approximate phone location, user's exact coordinates, and location information associated with media files, such as photos or videos, stored on the device. Only one app included a functionality that could utilize the `ACCESS_MEDIA_LOCATION`, but the other two did not. In most cases, `ACCESS_FINE_LOCATION` should suffice for applications that require precise location data for features like tracking exercise routes or providing location-based reminders. Using `ACCESS_COARSE_LOCATION` may be appropriate for less precise location needs, such as identifying the user's general area for weather updates or nearby points of interest. However, including multiple location tracking permissions without clear justification indicates overprivilege and potential privacy concerns.

Recommendations:

- Developers should run a static analysis (e.g., using MobSF) before each new update to identify possible security risks.
- Developers should consider adopting a permissions in-context approach, requesting permissions only when users initiate interactions with corresponding app features.
- Developers should disable non-essential permissions by default, enabling them only when necessary.

[2]Additional information about Android permissions: `https://developer.android.com/reference/android/Manifest.permission`

5.2 Apps' Security Testing & Coding

The results from the static analysis (section 4.2.2) revealed a large amount of hardcoded secrets and the use of weak cryptographic strategies such as ECB mode and CBC with PKCS5/PKCS7 padding. These vulnerabilities are well known and easy to discover with tools such as MobSF, which makes their prevalence in this study particularly concerning. It suggests that developers may not be abiding by the basic principles of secure coding and indicates a potential lack of maintenance of the app's security, raising concerns about the overall resilience of these applications against common threats and attacks.

One study found that, out of 30 apps examined, a majority of the apps opted for weak cryptographic methods, such as the ECB mode for data encryption and decryption [35]. However, this study only identified ECB mode being present in two apps ($n = 10$) while CBC with PKCS5/PKCS7 padding were present in half of the apps. Interestingly, none of the earlier studies of mHealth apps have mentioned findings of CBC with PKCS5/PKCS7 padding, indicating that this specific vulnerability might be less common or overlooked in previous research efforts. It is possible that previous studies have disregarded these findings as, while the use of CBC with PKCS5/PKCS7 padding indicates a potential problem, certain conditions must be met for this to be an issue. MobSF is also known to provide false positives for vulnerabilities, including those related to encryption configurations. A deeper analysis is therefore necessary to determine whether this should be classified as a vulnerability or a false positive.

Another study found that 80% of mHealth app developers had poor security knowledge and that 85% of developers had an inconsiderable security budget [1].

Given the substantial number of coding vulnerabilities uncovered in this study, including weak cryptographic practices, insecure cipher modes, hardcoded secrets, and app behavior logging, we believe this was also the case for the developers of the apps analyzed in this study. Otherwise, it would be hard to explain the existence of these common issues as the mitigation for it is not considerably hard nor time-consuming.

It is worth noting that during the code analysis (section 4.2.2), a large number of files were flagged by MobSF for the use of weak number generators and encryption strategies such as MD5 and SHA1. These files were scrutinized, disregarded, and classified as false positives, but as this was done manually, some may have been overlooked, potentially leaving exploitable vulnerabilities undiscovered.

Recommendations:

- Companies should allocate sufficient resources and budget for security testing.
- Companies should enforce secure coding standards and best practices across development teams to ensure consistency and adherence to security guidelines.
- Developers should implement automated security testing as part of the continuous integration and deployment pipeline to detect vulnerabilities early in the development lifecycle and prevent them from reaching production environments.

5.3 Insecure Communication

The results from the dynamic analysis provided positive insight into how user information was handled by the apps. All the apps used secure communication when transmitting sensitive user data, indicating good security practices. However, similar to earlier studies such as [25], three apps were found to reveal user email addresses or usernames as part of the RESTful URI request to an API endpoint. This practice is generally considered insecure because including email addresses or usernames in the RESTful URI request parameters exposes sensitive information directly in the request path. Although the contents of HTTPS requests, including paths and query parameters, are encrypted during transmission, this information can still be logged on the server side. Server logs could, therefore, contain sensitive user information, posing a risk to user privacy if the logs are accessed by unauthorized individuals or if proper log management practices are not followed. Additionally, URLs may be cached by browsers or stored in browser history, further increasing the risk of exposure.

Earlier studies, such as [53], have also found concerning trends regarding the security practices of mHealth apps. For instance, their investigation of over 20,000 mHealth apps revealed that as much as 45% of the investigated apps relied on unencrypted communication, with an additional 23% transmitting sensitive user information on insecure traffic. However, it's important to note that this study did not make any such observations, suggesting that the apps analyzed in this study exert good security practices in terms of secure communication protocols.

Recommendation:

- Developers should avoid including sensitive user information such as email addresses or usernames in GET requests, RESTful URIs or parameters. Instead, use non-identifiable attributes, such as an ID number or a hashed value.

5.4 Domains & Trackers

While our primary focus in this study was on analyzing the apps themselves, we found it important to include an analysis of the communicated domains and trackers discovered during the static and dynamic analysis. The results revealed some concerning information regarding the number of domains associated with certain apps. It was found that three apps communicated with more than 300 unique domains collectively (i.e., App10 ($n = 240$), App8 ($n = 236$), App4 ($n = 111$)). Although the nature of the data exchanged with each and every one of these domains was not scrutinized (i.e., unfeasible due to time limitations), the sheer volume of communication raises suspicion.

The lack of justification for such extensive interaction with external servers raises additional concerns about the behavior of these apps. We also find it suspicious that this extensive communication occurred during only 15 minutes of use. In contrast, the remaining apps maintained communication with a more reasonable number of domains, ranging from 0 to 34. This could be compared to the findings of [25], which found that, out of 27 apps, any app communicated with an average of 11.9 servers (min 1 - max 64). We also evaluated each domain TLS/SSL configuration, showing that 29% had some misconfiguration or fault, which resulted in a lower score (B 26.5%, T 2.5%) which is similar to the findings of [44]. There is no reason for domains to exhibit faulty TLS/SSL configurations, as these misconfigurations not only compromise the security of data transmission but also undermine user trust in the platform's overall security measures.

Furthermore, the prevalence of cross-device tracking raises concerns about how user data is shared and utilized for targeted advertising across various devices [48]. This practice not only compromises user privacy by potentially exposing their data and preferences but also makes it harder for users to control their data as it is not fully transparent which trackers gather what information. Moreover, trackers such as Tealium, Braze, and Amplitude, which collect, store, and create comprehensive user profiles, should not be allowed in apps that handle sensitive user data. Their prevalence also raises questions about the transparency and consent mechanisms in place for such data collection activities, as well as the potential for misuse of this data. The identification of Mixpanel's privacy breach clause, which denies users the right to delete their personal information, shines a light on the possible misuse of user data. This clause raises significant concerns about users' ability to control their personal data and underscores the need for greater transparency and accountability in tracker policies.

Recommendations:

- Developers should carefully evaluate the necessity of each domain communicated with by their apps and limit communication to only those essential for the app's functionality.
- Domain owners should prioritize the security of their TLS/SSL configurations to ensure the confidentiality and integrity of data transmitted between the app and the server.
- Developers should carefully consider the implications of using cross-platform advertisement trackers and minimize their usage wherever possible, as well as thoroughly review the privacy policies of third party trackers used in their apps to ensure compliance with privacy regulations and user rights.

5.5 Privacy Policies: Transparency, Availability, and Consent

All apps analyzed had a publicly available privacy policy link in their Google Play Store. Still, only eight apps had their privacy policy available during the sign-in and sign-up processes (i.e., in-app). These findings align with those of [52] and [25], which reported a correlation between higher availability of privacy policies and the app's popularity. This could be considered positive as it ensures that users have easy access to information regarding data handling practices before engaging with the app most of the time. The apps were also very transparent with what information might be gathered and what was shared with third parties. Furthermore, 9 out of 10 apps claimed to encrypt communication, yet all apps were found to do so.

Most apps (6/10) had a "contract by using" clause, meaning that consent to the privacy policies is given by a user just by using the service, which takes away the user's choice. Five apps also had "unilateral change," meaning the consent given for the privacy policy will not need to be renewed when changes are made to the policies. This can be compared to the findings of [25], which examined 27 mHealth apps and had the same percentile prevalence of the "contract by using" clause (59.2%, $n = 16$) and similar prevalence of the "unilateral change" clause (66.7%, $n = 18$). These clauses are extremely unfair as they allow the apps to decide for the user and possibly deprive them of the ability to make informed decisions about their data privacy. Because of the sensitive nature of these apps' data, they should rely on distinct informed consent.

Furthermore, the apps did not perform very well on the readability test, having an average score of 41.2 (ranging from 30-50), indicating the need for a higher level of education or for the reader to be at least 19-20 years old. This finding is similar to [25], which found that a majority of apps (89%, n = 27) scored between 30-50 on the readability index, whereas all the apps in this study scored within that range. While the

readability age might align with the intended target demographic for the apps, it raises concerns regarding accessibility and comprehension for younger users. The privacy policies should be readable to all users if the app has no age restriction.

Recommendations:

- Developers should make their privacy policies easily accessible, outlining the types of data collected, how it is used, and any third parties with whom it is shared.
- Developers should review and revise their privacy policies to possible unfair contract clauses such as “contract by using” and “unilateral change.”
- Privacy policies should be written in clear and understandable language, avoiding complex terms and words. Legal teams should aim for readability levels accessible to users of all ages and educational backgrounds to ensure that all users can make informed decisions about their data privacy.

5.6 Responsible Disclosure

The response rate from the companies following the responsible disclosure varied, with only one out of the approached stakeholders acknowledging the findings and providing a response. However, it’s important to recognize that the lack of response from the remaining companies does not necessarily indicate negligence or disregard for security issues. These companies are under no legal obligation to respond to responsible disclosure emails, although engagement with security researchers and ethical hackers can contribute to a more robust security posture.

5.7 Limitations

While this study provides valuable insights into health and fitness apps’ security and privacy landscape, several limitations should be considered. First, the scope of the study was limited to only ten free top-rated apps available on the Google Play Store, commonly used in four large English-speaking countries (USA, Canada, United Kingdom, and Australia). This selection criterion may not fully represent the diversity of global health and fitness apps, as most originate from North America. Additionally, it’s important to note that this project was time-limited, which impacted the sample size and restricted the number of apps that could be included in the study. Because of these limitations, this study focused on top-ranked health and fitness apps common in four countries with a rating of at least four stars.

Moreover, this study has relied heavily on MobSF for the static and dynamic analysis as it is well-documented, easy to use, and widely used in academia and industry. However, while MobSF is a robust framework composed of several internal tools, its use as the primary framework may limit the diversity of perspectives and insights gained.

Other frameworks and tools, such as Privado.ai[3] offer alternative approaches and Frida[4] which offers runtime monitoring and manipulation. These tools could provide complementary findings and perspectives to enhance the depth of future studies.

Furthermore, the server-side analysis relied solely on one AI tool, Qualys SSL, to assess the TLS/SSL configurations of the domains. While this tool is reputable, relying on one tool may overlook nuances or variations in server configurations that could impact security. The tool is also not time-efficient and might take up to ten minutes per execution.

In the privacy policy analysis, we relied on three tools: (1) WebFX for readability, (2) CLAUDETTE for identifying unfair clauses, and (3) GDPRWise for GDPR compliance. However, these AI-driven tools have their own set of limitations regarding accuracy. According to our experience, WebFX gave different scores depending on whether the URL for the privacy policy text or the copied text was entered. This might be because of the policy format, which might not be supported, or the link to the privacy policy contains additional external links. According to the developers, CLAUDETTE achieves an accuracy level of 78% in identifying unfair clauses, with a varying accuracy range of 74% to 95% in distinguishing between different categories of unfair clauses [29]. GDPRWises accuracy might vary depending on the same variables as WebFX [14], and one could speculate further, yet the accuracy of both GDPRWise and WebFX needs to be investigated further. Overall, it is necessary to consider these limitations when interpreting the findings.

[3]Privado's Open Source Privacy Code Scanning (https://www.privado.ai/open-source)
[4]Frida scripting tutorial (https://book.hacktricks.xyz/mobile-pentesting/android-app-pentesting/frida-tutorial)

Chapter 6

Conclusions

Health and fitness apps offer flexible ways to track your training progress, share it with friends, and find new exercises to improve your health at any time, anywhere. While these apps provide valuable tools for maintaining a healthy lifestyle, they also pose privacy risks that can compromise user data [25, 44, 52]. As technology advances, data privacy breaches have become increasingly prevalent, emphasizing the need for developers to prioritize data security and privacy in developing health and fitness apps. Thus, it is imperative for developers to adhere to data security and privacy best practices in the development of health and fitness apps, ensuring that user data remains secure and protected.

In this empirical study, we investigated the security and privacy risks associated with top-ranked mobile health and fitness apps for Android. Our study found a high prevalence of vulnerabilities related to sub-optimal coding practices (10/10 apps), hardcoding sensitive information (10/10 apps), and using known insecure encryption configurations (5/10 apps). Further threats related to misconfigurations and overprivilege allowed apps to be installed on older vulnerable Android versions and acquire permissions reaching outside of the apps' intended use. The apps were also found to engage with various questionable trackers, expose sensitive data in HTTP(s) traffic, log user actions and POST values, and frequent extensive domain communication having inadequate TLS/SSL configurations.

In conclusion, this study sheds light on the prevalent vulnerabilities in health and fitness apps, finding that even top-ranked apps are prone to common issues, which may potentially impact millions of users. Nonetheless, compared to prior work in the field, the tests conducted in this study yielded more positive results overall. This suggests that developers of the analyzed apps are better informed and more proficient in addressing many of the privacy and security risks that are usually found. However, the extensive domain communication observed in three apps during the brief 15-minute usage period as well as the numerous misconfigurations and sub-optimal coding practices, are seen as careless and unnecessary and should be addressed promptly. While some vulnerabilities may not pose immediate severe threats, developers should

not overlook even minor ones and aim for a zero-threat environment. Moving forward, conducting additional research with larger sample sizes and extended timeframes (e.g., longitudinal testing) would enable a more comprehensive understanding of health and fitness apps' overall security and privacy posture.

6.1 Future Work

As previously stated, some limitations in this study refer to the sample size and time spent to classify the health and fitness apps. Therefore, for future research, we encourage expanding the datasets to cover a broader range of health and fitness apps to provide more comprehensive insights into the prevalence of vulnerabilities and potential threats. Additionally, incorporating other tools, such as Privado's privacy code scanning and Frida scripting, could enhance the depth of analysis, allowing for a more nuanced examination of app behaviors and security measures. It would also be interesting to evaluate the apps based on the OWASP mobile top 10 and test apps for each vulnerability to assess their prevalence within mHealth apps.

Future research might also want to conduct studies over extended periods to observe how data sharing and access permissions evolve longitudinally (e.g., [15, 36]). Considering the large number of domains the studied apps were communicating with, it would be especially interesting to investigate the patterns of data exchange between apps and these domains, as well as any changes in privacy practices over time.

Bibliography

[1] Aljedaani, Bakheet, Ahmad, Aakash, Zahedi, Mansooreh, and Babar, M Ali. "An empirical study on developing secure mobile health apps: The developers' perspective". In: *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2020, pp. 208–217.

[2] Asemi, Hadi. "A Study on API Security Pentesting". Master of Science in Computer Science. California Polytechnic State University, San Luis Obispo, June 2023.

[3] Baehman, Christopher D and Hexmoor, Henry. "Smartphone App Security Risks". In: (2019).

[4] Baitha, Anuj Kumar and Vinod, Smitha. "Session hijacking and prevention technique". In: *Int. J. Eng. Technol* 7.2.6 (2018), pp. 193–198.

[5] Bishop, Matt. *Introduction to computer security*. Addison-Wesley Professional, 2004.

[6] Chen, Xin, Yu, Haihua, Yu, Dongjin, Chen, Jie, and Sun, Xiaoxiao. "Predicting Android malware combining permissions and API call sequences". In: *Software Quality Journal* 31.3 (2023), pp. 655–685.

[7] Chow, Emily. "Ethical hacking & penetration testing". In: *University of Waterloo, Waterloo, Canada, No. AC* 626 (2011).

[8] *Data protection basics*. European Data Protection Board (EDPB). 2024. URL: `https://edpb.europa.eu/sme-data-protection-guide/home_en` (visited on 01/30/2024).

[9] Developers, Android. *Android Developer Documentation: App permissions overview*. 2024. URL: `https://developer.android.com/guide/topics/permissions/overview#:~:text=App%20permissions%20build%20on%20system%20security%20features%20and,specific%20task%20or%20action%20that%20the%20user%20invokes`.

[10] Egele, Manuel, Brumley, David, Fratantonio, Yanick, and Kruegel, Christopher. "An empirical study of cryptographic misuse in Android applications". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 73–84.

[11] Firebase. *API keys | Firebase Documentation*. Accessed: 2024-06-12. 2024. URL: https://firebase.google.com/docs/projects/api-keys.

[12] Google. *Google API Keys Best Practices*. Accessed: April 2024. Google. n.d. URL: https://support.google.com/googleapi/answer/6310037?hl=en.

[13] *Google Support: Remediation for Unsafe Encryption Mode Usage*. https://support.google.com/faqs/answer/10046138?hl=en. Accessed: April 2024.

[14] Guo, Hanyang, Dai, Hong-Ning, Luo, Xiapu, Zheng, Zibin, Xu, Gengyang, and He, Fengliang. "An Empirical Study on Oculus Virtual Reality Applications: Security and Privacy Perspectives". In: *arXiv preprint arXiv:2402.13815* (2024).

[15] Hatamian, Majid, Momen, Nurul, Fritsch, Lothar, and Rannenberg, Kai. "A multilateral privacy impact analysis method for android apps". In: *Privacy Technologies and Policy: 7th Annual Privacy Forum, APF 2019, Rome, Italy, June 13–14, 2019, Proceedings 7*. Springer. 2019, pp. 87–106.

[16] He, Dongjing. "Security threats to Android apps". PhD thesis. University of Illinois at Urbana-Champaign, 2014.

[17] He, Dongjing, Naveed, Muhammad, Gunter, Carl A, and Nahrstedt, Klara. "Security concerns in Android mHealth apps". In: *AMIA annual symposium proceedings*. Vol. 2014. American Medical Informatics Association. 2014, p. 645.

[18] Hussain, Muzammil, Al-Haiqi, Ahmed, Zaidan, Aws Alaa, Zaidan, Bilal Bahaa, Kiah, M, Iqbal, Salman, Iqbal, Shaukat, and Abdulnabi, Mohamed. "A security framework for mHealth apps on Android platform". In: *Computers & Security* 75 (2018), pp. 191–217.

[19] Integritetsskyddsmyndigheten (IMY). *Grundläggande principer*. 2024. URL: https://www.imy.se/verksamhet/dataskydd/det-har-galler-enligt-gdpr/grundlaggande-principer/ (visited on 02/23/2024).

[20] Integritetsskyddsmyndigheten (IMY). *Så hänger lagarna ihop*. 2024. URL: https://www.imy.se/verksamhet/dataskydd/ (visited on 01/30/2024).

[21] InternetPrivacy. *What is Internet Privacy?* 2024. URL: https://www.internetprivacy.com/what-is-internet-privacy/ (visited on 02/23/2024).

[22] Invisibly. *Data Privacy Principles*. 2024. URL: https://www.invisibly.com/learn-blog/data-privacy-principles/ (visited on 02/23/2024).

[23] Iwaya, Leonardo Horn. "Engineering privacy for mobile health data collection systems in the primary care". Universitetstryckeriet, Karlstad 2018. Doctoral Thesis. SE-651 88 Karlstad, Sweden: Faculty of Health, Science, Technology, Department of Mathematics, and Computer Science, 2019.

[24] Iwaya, Leonardo Horn, Ahmad, Aakash, and Babar, M Ali. "Security and privacy for mHealth and uHealth systems: a systematic mapping study". In: *IEEE Access* 8 (2020), pp. 150081–150112.

[25] Iwaya, Leonardo Horn, Babar, M Ali, Rashid, Awais, and Wijayarathna, Chamila. "On the privacy of mental health apps: An empirical investigation and its implications for app development". In: *Empirical Software Engineering* 28.1 (2023), p. 2.

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

[27] Kotipalli, Srinivasa Rao and Imran, Mohammed A. *Hacking Android*. Packt Publishing Ltd, 2016.

[28] Lab, Software. *What is Internet Security?* 2024. URL: https://softwarelab.org/blog/what-is-internet-security/ (visited on 02/23/2024).

[29] Lippi, Marco, Pałka, Przemysław, Contissa, Giuseppe, Lagioia, Francesca, Micklitz, Hans-Wolfgang, Sartor, Giovanni, and Torroni, Paolo. "CLAUDETTE: an automated detector of potentially unfair clauses in online terms of service". In: *Artificial Intelligence and Law* 27 (2019), pp. 117–139.

[30] "Chapter 3 - An Introduction To Cryptography". In: *Next Generation SSH2 Implementation*. Ed. by Dale Liu, Max Caceres, Tim Robichaux, Dario V. Forte, Eric S. Seagren, Devin L. Ganger, Brad Smith, Wipul Jayawickrama, Christopher Stokes, and Jan Kanclirz. Burlington: Syngress, 2009, pp. 41–64. ISBN: 978-1-59749-283-6. DOI: https://doi.org/10.1016/B978-1-59749-283-6.00003-9. URL: https://www.sciencedirect.com/science/article/pii/B9781597492836000039 (visited on 04/03/2024).

[31] Liu, Kaijun, Xu, Guoai, Zhang, Xiaomei, Xu, Guosheng, Zhao, Zhangjie, et al. "Evaluating the privacy policy of android apps: A privacy policy compliance study for popular apps in china and europe". In: *Scientific Programming* 2022 (2022).

[32] Liu, Ru, Menhas, Rashid, Dai, Jianhui, Saqib, Zulkaif Ahmed, and Peng, Xiang. "Fitness apps, live streaming workout classes, and virtual reality fitness for physical activity during the COVID-19 lockdown: an empirical study". In: *Frontiers in public health* 10 (2022), p. 852311.

[33] Madrid, Universidad Politécnica de. *Logcat Mining: Execution Logs in Android*. https://blogs.upm.es/sse/2020/10/27/logcat-mining-execution-logs-in-android/. Accessed: April 12, 2024. 2020.

[34] *Manifest.permission*. URL: https://developer.android.com/reference/android/Manifest.permission (visited on 04/08/2024).

[35] Mia, Md Raihan, Shahriar, Hossain, Valero, Maria, Sakib, Nazmus, Saha, Bilash, Barek, Md Abdul, Faruk, Md Jobair Hossain, Goodman, Ben, Khan, Rumi Ahmed, and Ahamed, Sheikh Iqbal. "A comparative study on HIPAA technical safeguards assessment of android mhealth applications". In: *Smart Health* 26 (2022), p. 100349.

[36] Momen, Nurul. "Measuring Apps' Privacy-Friendliness: Introducing transparency to apps' data access behavior". PhD thesis. Karlstad University, 2020.

[37] Open Web Application Security Project (OWASP). *OWASP Mobile Top 10: Insecure Communication*. Accessed: April 3, 2024. 2023. URL: https://owasp.org/www-project-mobile-top-10/2023-risks/m5-insecure-communication.html.

[38] Open Web Application Security Project (OWASP). *OWASP Web Security Testing Guide: Testing for Padding Oracle*. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/02-Testing_for_Padding_Oracle. Accessed: April 3, 2024. 2024.

[39] OWASP. *Mobile Top 10 - M8: Security Misconfiguration*. Accessed: April 2024. OWASP. 2023. URL: https://owasp.org/www-project-mobile-top-10/2023-risks/m8-security-misconfiguration.html.

[40] OWASP. *OWASP Mobile Top 10: Insufficient Binary Protection*. Accessed: April 2024. OWASP. 2023. URL: https://owasp.org/www-project-mobile-top-10/2023-risks/m7-insufficient-binary-protection.html.

[41] OWASP. *OWASP Web Security Testing Guide: Testing for Weak Encryption*. Accessed: April 2024. OWASP, 2022. URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption.

[42] OWASP. *Password Plaintext Storage*. https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage. Accessed: April 2, 2024. n.d.

[43] OWASP. *Secrets Management Cheat Sheet*. Accessed: April 2024. OWASP. 2022. URL: https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html.

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

[45] Ranganath, Venkatesh-Prasad and Mitra, Joydeep. "Are free android app security analysis tools effective in detecting known vulnerabilities?" In: *Empirical Software Engineering* 25 (2020), pp. 178–219.

[46] Raymond, Anjanette, Schubauer, Jonathan, and Madappa, Dhruv. "Over-Privileged Permissions: Using Technology and Design to Create Legal Compliance". In: *Journal of Business and Technology* 15.2020 (2020).

[47] Sahare, Bhawana, Naik, Ankit, and Khandey, Shashikala. "Study of ethical hacking". In: *Int. J. Comput. Sci. Trends Technol* 2.4 (2014), pp. 6–10.

[48] Solomos, Konstantinos, Ilia, Panagiotis, Ioannidis, Sotiris, and Kourtellis, Nicolas. “Talon: An Automated Framework for Cross-Device Tracking Detection”. In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 2019, pp. 227–241.

[49] Statista. *Downloads of leading fitness and workout mobile apps worldwide in the month of January from 2017 to 2024*. 2024. URL: https://www.statista.com/statistics/1239806/growth-top-fitness-mobile-apps-downloads/ (visited on 03/19/2024).

[50] Statista. *Number of health apps available in the Apple App Store worldwide from 1st quarter 2015 to 2nd quarter 2021*. 2023. URL: https://www.statista.com/statistics/779910/health-apps-available-ios-worldwide/ (visited on 02/02/2024).

[51] Statista. *Number of health apps available in the Google Play Store worldwide from 1st quarter 2015 to 2nd quarter 2021*. 2023. URL: https://www.statista.com/statistics/779919/health-apps-available-google-play-worldwide/ (visited on 02/02/2024).

[52] Tangari, Gioacchino, Ikram, Muhammad, Ijaz, Kiran, Kaafar, Mohamed Ali, and Berkovsky, Shlomo. *Mobile health and privacy: cross sectional study. BMJ 373: n1248*. 2021.

[53] Tangari, Gioacchino, Ikram, Muhammad, Sentana, I Wayan Budi, Ijaz, Kiran, Kaafar, Mohamed Ali, and Berkovsky, Shlomo. “Analyzing security issues of Android mobile health and medical applications”. In: *Journal of the American Medical Informatics Association* 28.10 (2021), pp. 2074–2084.

[54] Verderame, Luca, Caputo, Davide, Romdhana, Andrea, and Merlo, Alessio. “On the (un) reliability of privacy policies in android apps”. In: *2020 international joint conference on neural networks (IJCNN)*. IEEE. 2020, pp. 1–9.

[55] Westin, Alan F. “Privacy and freedom”. In: *Washington and Lee Law Review* 25.1 (1968), p. 166.

[56] Zaeem, Razieh Nokhbeh, German, Rachel L, and Barber, K Suzanne. “Privacycheck: Automatic summarization of privacy policies using data mining”. In: *ACM Transactions on Internet Technology (TOIT)* 18.4 (2018), pp. 1–18.

[57] Zhang, Yueheng, Shu, Junliang, Zhang, Yuanyuan, Li, Juanru, Wang, Qing, and Gu, Dawu. “An Empirical Study of Insecure Communication in Android Apps”. In: *DEStech Transactions on Computer Science and Engineering* (2017).

[58] Zimmeck, Sebastian, Wang, Ziqi, Zou, Lieyong, Iyengar, Roger, Liu, Bin, Schaub, Florian, Wilson, Shomir, Sadeh, Norman, Bellovin, Steven, and Reidenberg, Joel. “Automated analysis of privacy requirements for mobile apps”. In: *2016 AAAI Fall Symposium Series*. 2016.