ABSTRACT

ANDROID SECURITY: ANALYSIS AND APPLICATIONS

by
Raina Samuel

The Android mobile system is home to millions of apps that offer a wide range of functionalities. Users rely on Android apps in various facets of daily life, including critical, e.g., medical, settings. Generally, users trust that apps perform their stated purpose safely and accurately. However, despite the platform's efforts to maintain a safe environment, apps routinely manage to evade scrutiny. This dissertation analyzes Android app behavior and has revealed several weakness: lapses in device authentication schemes, deceptive practices such as apps covering their traces, as well as behavioral and descriptive inaccuracies in medical apps. Examining a large corpus of applications has revealed that suspicious behavior is often the result of lax oversight, and can occur without an explicit intent to harm users. Nevertheless, flawed app behavior is present, and is especially problematic in apps that perform critical tasks. Additionally, manufacturer's and app developer's claims often do not mirror actual functionalities, e.g., as we reveal in our study of LG's Knock Code authentication scheme, and as evidenced by the removal of Google Play medical apps due to overstated functionality claims. This dissertation makes the following contributions: (1) quantifying the security of LG's Knock Code authentication method, (2) defining deceptive practices of self-hiding app behavior found in popular apps, (3) verifying abuses of device administrator features, (4) characterizing the medical app landscape found on Google Play, (5) detailing the claimed behaviors and conditions of medical apps using ICD codes and app descriptions, (6) verifying errors in medical score calculator app implementations, and (7) discerning how medical apps should be regulated within the jurisdiction of regulatory frameworks based on their behavior and data acquired from users.

ANDROID SECURITY: ANALYSIS AND APPLICATIONS

by
Raina Samuel

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology and
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Information Systems

Department of Informatics

December 2022

APPROVAL PAGE

ANDROID SECURITY: ANALYSIS AND APPLICATIONS

Raina Samuel

Iulian Neamtiu, Dissertation Advisor Date
Professor, Computer Science, NJIT

Michael J. Lee, Committee Member Date
Associate Professor, Informatics, NJIT

Salam Daher, Committee Member Date
Assistant Professor, Informatics, NJIT

Tomer Weiss, Committee Member Date
Assistant Professor, Informatics, NJIT

Zhiyong Shan, Committee Member Date
Assistant Professor, School of Computing,
Wichita State University, Wichita, Kansas

BIOGRAPHICAL SKETCH

Author: Raina Samuel

Degree: Doctor of Philosophy

Date: December 2022

Undergraduate and Graduate Education:

- Masters of Science in Software Engineering,
  New Jersey Institute of Technology, 2018
- Bachelor of Science in Computer Information Technology,
  Southern New Hampshire University, 2016

Major: Information Systems

Presentations and Publications:

Raina Samuel, Iulian Neamtiu, Sydur Rahaman "Could Medical Apps Keep Their Promises?," *E-Health '22: 16th Multi Conference on Computer Science and Information Systems*, Pages 173-180, July 2022.

Raina Samuel, Iulian Neamtiu, Sydur Rahaman, James Geller "Characterizing Medical Android Apps," *E-Health '22: 16th Multi Conference on Computer Science and Information Systems*, Pages 155-162, July 2022.

Sydur Rahaman, **Raina Samuel**, Iulian Neamtiu "Quantifying Nondeterminism and Inconsistency in Self-organizing Map Implementations," *AITest '2I: 2021 IEEE International Conference on Artificial Intelligence*, Pages 85-92, August 2021.

Raina Samuel, Philipp Markert, Adam J. Aviv, Iulian Neamtiu "Knock, Knock. Who's There? On the Security of LG's Knock Codes," *SOUPS '20: Proceedings of the 16th Symposium on Usable Privacy and Security*, Pages 37-60, August 2020.

Zhiyong Shan, **Raina Samuel**, Iulian Neamtiu "Device Administrator Use and Abuse in Android: Detection and Characterization," *Mobicom '19: The 25th Annual International Conference on Mobile Computing and Networking*, Article No.: 51, Pages 1–16, October 2019.

Zhiyong Shan, Iulian Neamtiu, **Raina Samuel** "Self-Hiding Behavior in Android Apps: Detection and Characterization," *ICSE '18: Proceedings of the 40th International Conference on Software Engineering*, Pages 728–739, May 2018.

To my family and my loved ones:
Amma, Anicha, Roy, Renee, Simon, Christine,
Patrick, Jimmy, Alice, Deli, Will, Enrique,
Francis, Fidel, Mahsa, Carol, Steven,
Leo (the sweetest orange tabby),
and Gus (the loudest black kitty)

ACKNOWLEDGMENT

I would like to begin by thanking Dr. Iulian Neamtiu, though it seems impossible to fully express my gratitude and appreciation for all he had done for me. My initial foray into research was piqued by a special topics mobile security course I had taken under him during my Master's. I had no intention to pursue a PhD at the time, however since I did well in the course and was extremely interested with the material that changed very quickly. With Professor Neamtiu's guidance and support, I was able to be part of many research efforts and projects and had the opportunity to attend and present at many conferences. I learned so much, not just from the topics I was introduced to, but also aspects of research, such as presenting and writing (and Latex tricks), as well as teaching and dealing with students. I will look fondly back at those 8AM submission deadlines and all-nighters(and possibly continue them) to strive for perfection in our submissions. Thank you so much Professor for your patience and your encouragement pushing me and challenging me to test my abilities and achieve my full potential!

I sincerely thank the members of my committee: Dr. Michael J. Lee, Dr. Tomer Weiss, and Dr. Salam Daher. I thank Dr.Lee for his help, guidance, and great efforts to answer my questions regarding program requirements as well as providing meaningful feedback. I appreciate Dr. Weiss and Dr. Daher for their interest and for having an open mind to be part of my committee, even though these are topics that are not related to their respective fields.

I also thank Dr. Zhiyong Shan for not only being part of my committee but for also including me and guiding me in our publications together. Being a co-author and working with Dr. Shan enabled me to thoroughly learn various aspects and vulnerabilities found in Android apps and the Android Operating System. Working with Dr. Adam Aviv was a very rewarding experience. Our collaboration spurred from a question I had asked after his talk regarding mobile authentication security. I am so grateful that I was included in this research and I was experienced user based research for the first time and learned various aspects when dealing with human responses and subjects. I thank Dr. James Geller for his guidance and collaboration for our medical apps characterization paper. That independent study led to many other interesting topics and research discussed and presented in this dissertation.

I truly thank Sydur Rahaman for all of his help, advice, and opinions, not only as a co-author but also as a lab colleague and friend. Despite being a fan of Real Madrid, Sydur is the ideal teammate to collaborate with on any project, as well as being a great conference buddy. I also thank both Xin Yin and Muyeed Ahmed for being great lab colleagues throughout the years.

I thank my family, especially my mom, Selene, and dad, Samuel, for their constant encouragement and support throughout my entire life. Their hard work and sacrifices helped me achieve what I was able to accomplish finally after many years of work. I thank my younger brother Roy, for deciding to attend NJIT for his undergrad, which led me to follow him for my Master's, but also for his feedback and being a great sounding board for research ideas and topics (and also giving me crash courses in statistics). I'm so thankful for my quarantine, remote learning buddy and my little sister, Renee. Her emotional support especially during the more tumultuous parts of this whole process helped keep me sane and motivated.

Finally, I want to thank all my friends and everyone else who had helped make this dissertation possible. It's hard to mention everyone here, but know that I truly am grateful for everything everyone has done for me to allow me to be here today.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Mobile devices are an ubiquitous and irreplaceable aspect of daily life. Therefore, it is essential to understand and maintain a safe user friendly environment.

1.1 Dissertation Scope

The scope of this dissertation is to find areas of vulnerability in the Android mobile ecosystem while also providing a means of fixing such issues. We focus our research on the following key points of interest: observing deceptive app behavior, permission abuse in apps, weak authentication schemes for unlocking mobile devices, characterizing the range of medical apps which perform critical tasks, uncovering inaccuracies and inconsistencies in medical calculator apps, and understanding the scope of regulatory frameworks in approving medical apps.

1.2 Problem Description

Mobile security is a broad field that addresses different issues in mobile platforms, or in this specific study, the Android mobile platform. We primarily focus on a specific recurring problem in this dissertation: app behavior is not apparent to the user. Users trust applications to perform their stated uses without suspecting any type of suspicious behavior. Additionally, suspicious behavior is not limited to malicious applications, rather it can occur in popular benign trusted applications, intentionally or unintentionally.

1.3 Dissertation Objectives

The objective of this dissertation is to reveal and address vulnerabilities found in the Android ecosystem. We will discuss issues that arise in mobile authentication

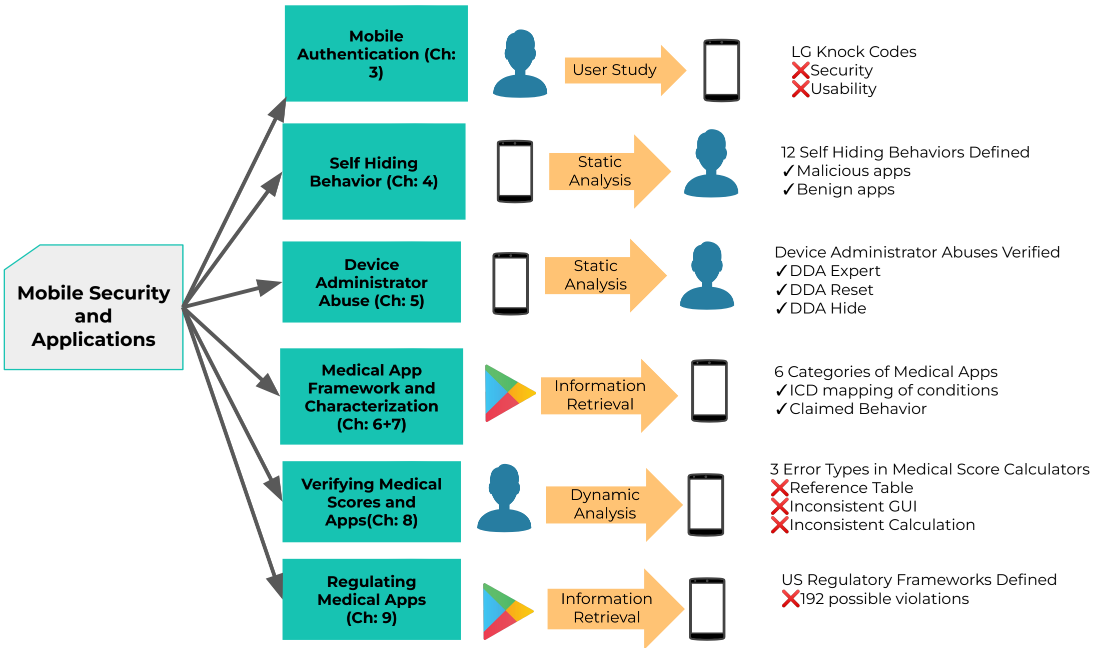

Mobile Security and Applications
Mobile Authentication (Ch: 3)
User Study
LG Knock Codes
❌Security
❌Usability
Self Hiding Behavior (Ch: 4)
Static Analysis
12 Self Hiding Behaviors Defined
✓Malicious apps
✓Benign apps
Device Administrator Abuse (Ch: 5)
Static Analysis
Device Administrator Abuses Verified
✓DDA Expert
✓DDA Reset
✓DDA Hide
Medical App Framework and Characterization (Ch: 6+7)
Information Retrieval
6 Categories of Medical Apps
✓ICD mapping of conditions
✓Claimed Behavior
Verifying Medical Scores and Apps(Ch: 8)
Dynamic Analysis
3 Error Types in Medical Score Calculators
❌Reference Table
❌Inconsistent GUI
❌Inconsistent Calculation
Regulating Medical Apps (Ch: 9)
Information Retrieval
US Regulatory Frameworks Defined
❌192 possible violations

Figure 1.1 Dissertation overview.

schemes, permission abuses and self hiding behavior found among apps. We will also present a classification scheme to better understand medical apps and their purported behaviors. Our intended research is to delve further in Android security, focusing more on medical app data management, accuracy, and general app data tracing. We provide a visualization of this dissertation in Figure 1.1.

1.4 Dissertation Organization

We begin in Chapter 2 by detailing popular mobile systems and more specifically, the Android ecosystem. We discuss the software architecture and security of the Android operating system, while also discussing exploits and vulnerabilities that have been found previously by other researchers.

LG Knock Code Security and Usability. To examine mobile security, the first aspect to discuss is device unlock methods. In Chapter 3, we present a study regarding security and usability flaws in mobile authentication schemes, specifically LG's Knock

Codes. We detail our user study and results regarding this recently-introduced authentication method and ultimately discover it is actually not as secure as LG claims and it suffers with usability.

Self Hiding Behavior. We then move on to describe and define deceptive app behavior in the case of self-hiding behavior in Chapter 4. We explain our study which both defines and detects this behavior not only in malicious applications, which is expected, but also in popular benign trusted apps.

Device Administrator Abuse. Another aspect of deceptive app behavior is found in device administrator abuses. In Chapter 5, we study this feature found on mobile devices that was implemented to provide capabilities for companies to control mobile fleets given to employees, such as locking and wiping devices remotely. We discover that such capabilities have been exploited and leveraged to cause critical issues, such as being unable to uninstall certain device administrator apps. We characterize this behavior and provide a method of detecting device administrator abuse.

Characterizing Medical Applications. App security requires an understanding the app landscape. In Chapter 6, we provide an automatic characterization scheme to better identify and understand the functionality of medical applications on Google Play.

Claimed Behavior of Medical Apps. We look into medical app claims and discover how many apps overstate their actual functionality in Chapter 7. We map app behavior with ICD-11 conditions to observe their possible functionalities. We reveal that app claims are not reliable and the need for better regulation and scrutiny of medical apps.

Verifying Medical Calculator Apps. Additionally, we verify the reliability of medical calculators and their respective reference tables in Chapter 8. We find inconsistencies and inaccuracies in many medical calculators which calculate critical scores in hospital settings. These errors could result in severe consequences towards patients. Many of these errors are a product of incorrect references and specifications as well as developer errors.

Regulating Medical Apps. In Chapter 9, we discuss the various regulations set in place for medical apps. We specifically discuss regulations based in the US and uncover applications that may need to be scrutinized by the FTC due to dubious claims.

CHAPTER 2

BACKGROUND

To begin our discussion of mobile systems, we provide a brief background on their inception, popularity, and software architecture. We also introduce examples of mobile security issues and vulnerabilities.

2.1 Mobile Systems

In 1992, IBM created the first smartphone with a touchscreen, ‘The Simon Personal Communicator (SPC)’. SPC was released to the public in 1994 and sold roughly 50,000 units [28]. While the SPC was ahead of its time by being the first providing certain functionalities, such as predictive typing, sending emails and faxes, it was short lived. Eventually other iterations of cellular devices or “smart phones” began to take hold, spearheaded by Nokia with the Nokia 3310 [91] as well as the dominant mobile OS at the time, Symbian [57]. However, despite these advances in mobile technology, it was in 2007 when Apple released the first iPhone and spurred the mobile industry that persists to this day [5]. The iPhone was the first smartphone to be solely touch-based in its interface, removing most physical buttons found previously. Shortly after in 2008, as its answer to the iPhone, Google released the T-Mobile G1, also known as the HTC Dream, marking the introduction of Android 1 to the mobile marketplace [14]. Two years later in 2010, Microsoft attempted to break through the consumer smartphone market with the Windows Phone. The Windows Phone served as a successor to Windows Mobile, which was a popular but discontinued mobile OS that was aimed for the enterprise market in the early 2000s. Ultimately, Windows Phone met the same demise and was discontinued in 2017 [120]. Overall, since 2009, the mobile OS market share has changed drastically; with initially dominant systems such as Symbian and Sony Ericsson fading away and with Android rising and ultimately replacing them.

2.2 Mobile Markets and Apps

With the popularity of smartphones, mobile apps have also grown in use. Mobile apps are programs designed to run on smartphones and are designed to aid with productivity, to provide assistance, or to be sources of entertainment and communication. Users are able to download apps for free or for a fee on an app marketplace (e.g., Apple App Store or Google Play). Apps are designed to work on specific platforms, meaning that an app designed for iOS cannot be used on an Android device, and vice versa. We will discuss next the features of the two most popular app marketplaces as well as an overview of popular apps and uses.

iOS. Released in 2007 with the first iPhone, iOS is the operating system used by Apple for iPhones. After Android, it is the second most popular mobile OS worldwide, accounting for 28.3% of users as of November 2022 [92]. Users are able to install and use third party apps on iOS by installing them from Apple's App Store, a digital app marketplace where apps are curated and vetted via security checks prior to being offered to users. In 2008, Apple released the iOS Software Development Kit (SDK), allowing for third-party apps to be developed and published on the App Store [131]. Initially starting with 500 apps in 2008, as of November 2022, iOS has 1.6 million apps available for download on the App Store [24].

Android. In 2008 Android became the world's most popular mobile OS (with 73% of the total market share at that time [22]). Developed by Google, Android is open source, with its source code available by the Android Open Source Project, which is customized by original equipment manufacturers (OEMs). A reason for Android's widespread popularity is because it is not solely tied to Google made devices, while iOS is limited to only Apple iPhones. Rather, Android dominates the marketplace because OEMs can use Android as their device's OS, thanks to the open source nature of the platform. Android users can download apps via Google Play, Android's app marketplace. As of November 2022, there are 3.55 million apps on Google Play [24]. Apps have grown in number on Google Play as Android had grown in popularity until December 2017 where almost 700,000 apps were removed [115]. Overall, Google has been cracking down on apps that mimic other apps, apps with inappropriate content, and apps that are potentially harmful. Additionally, apps can be downloaded from third party markets as well such as F-Droid [7] or OEM based stores like the Samsung Galaxy Store [15] and others. The Android ecosystem has been affected by many vulnerabilities, which has prompted Google to take more steps ensuring security and safety for users over the years. We will further discuss the evolution of mobile security in Android later in this chapter.

Mobile Apps. Mobile app usage has increased significantly over the years. The average smartphone user spends three hours a day on their device, using on average 10 apps a day, while having over 80 apps installed on their device [11]. Initially, apps were used for general productivity or information retrieval uses, such as email clients, web browsers, or weather apps. However with growing public demands, top apps being downloaded are social media apps such as TikTok, Facebook, and Instagram [122]. Mobile apps are not limited to leisure or entertainment; rather, they can be used for a variety of critical tasks. There is a surge of mobile e-commerce apps and more users find mobile shopping to be more convenient [88]. Along with e-commerce, mobile banking has been a growing category of apps, with 86.5% of Americans having used a mobile device to check their bank balance [129]. Moreover, medical health apps have grown in popularity while able to perform a range of critical tasks, handling sensitive information. With the varied uses of mobile apps across platforms, it is imperative

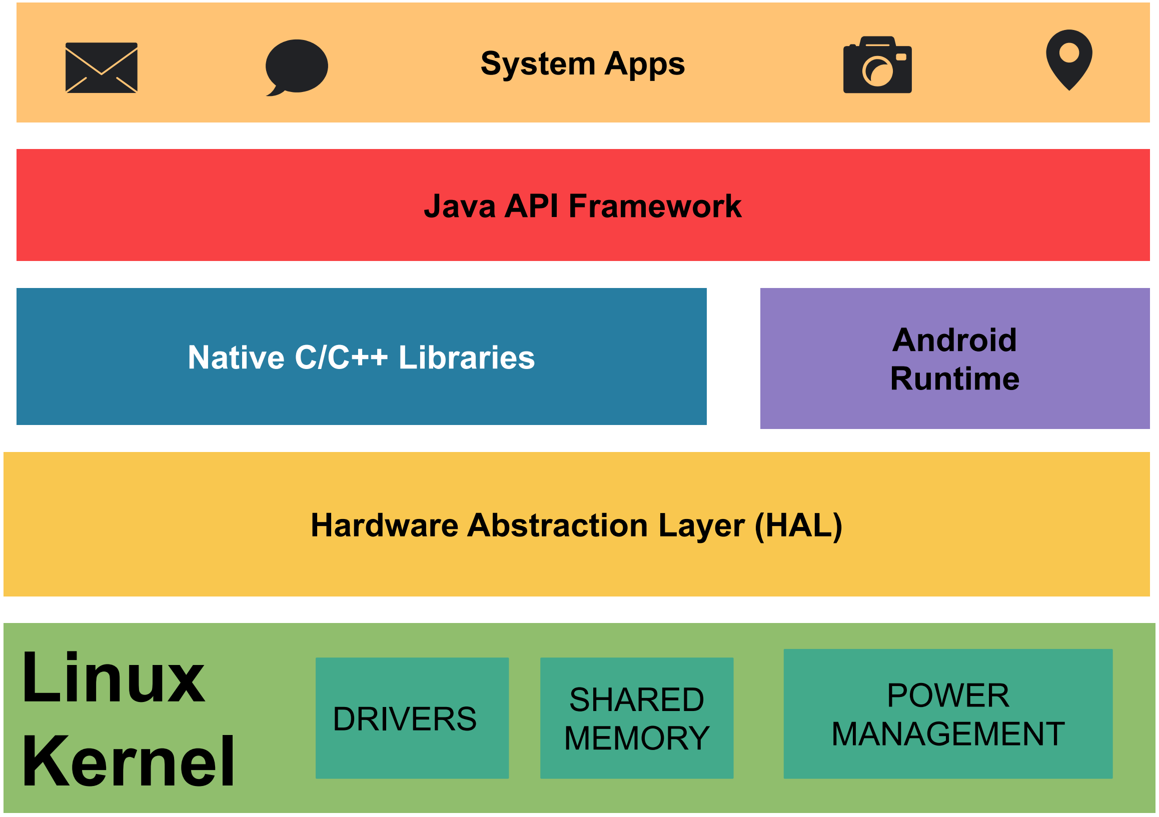


Figure 2.1 Visualization of the Android software architecture.

that mobile security be taken seriously. User security and privacy are increasingly at risk due to the prevalence of mobile systems. We will next look into mobile app development to better understand app design and app capabilities.

2.3 Android App Development

We will now discuss the development and structure of an Android app.

2.3.1 Software Stack

Before we explore app development, we first examine the platform architecture. We provide a visualization of the Android platform architecture in Figure 2.1. Android devices use a Linux kernel, but due to the variety of devices, the kernel version changes based on the device model and make [80]. The kernel handles requests from the software to the hardware. Nevertheless, despite having a Linux kernel foundation, Android has few similarities with the standard desktop Linux distributions [123]. On top of the Linux kernel, there is the hardware abstraction layer or HAL. The HAL contains library modules which interface with specific hardware components, such as the camera and Bluetooth. The HAL serves as a bridge between device hardware capabilities and higher level Java API libraries [13]. The next layer includes native C/C++ libraries and Android Runtime (ART). ART is a runtime environment that uses ahead of time compilation (AOT), meaning that it precompiles app bytecode into native (binary) code when an app is first installed. In doing so, app execution is much faster [142]. Before using ART in version 5.0, Android was using Dalvik Virtual Machines as the app runtime environment. Dalvik VMs relied on a just in time compiler (JIT) for interpreting bytecode and overall was inefficient, providing a lot of overhead. Both ART and HAL, along with other system components, require native C/C++ libraries since they are built from native code [13]. To expose some of these native libraries' functionalities to apps, Android provides Java framework APIs. The Java API framework simplifies the use of modular components and provides content providers, managers, and a view system for apps. Finally, on top of all the stack layers are system apps, or apps that come preinstalled and usually cannot be uninstalled by the user.

2.3.2 App Components

Next we will delve into the basis of a standard Android app. Android apps can be written in Java, Kotlin, and C++. They are then compiled along with other resource files to create an 'APK' or an Android package. There are four key components of an app: activities, services, broadcast receivers, and content providers.

Activities. An activity allows users to interface with the app, providing a way for the app to implement user flows between each facet of the app and allows the system to coordinate what the user is prioritizing when using the app. Activities are managed in stacks; new stacks are placed on the current stack and becomes the running activity. An activity has four states:

- Active or running (User is currently interacting with the activity).
- Visible (Activity is not fully in focus but still presented).
- Stopped or hidden (Activity is hidden by another activity).
- Destroyed (Process is killed).

An activity's lifecycle begins with a call to onCreate() and is ended with the onDestroy() method [1].

Services. A service helps keep an app run in the background in order to perform long running operations or to work with remote processes. Services are entry points to keep apps running in the background. They tell the system how to handle an app [16]. There are three types of services:

- *Foreground services* perform noticeable operations to the user. They must display a notification to the user that it is running even when they are not interacting with the app.
- *Background services* perform unnoticeable operations to the user.
- *Bound services* provide a client-server interface for components to interact with the service and perform a variety of different tasks. When an app component unbinds from the service, the service is destroyed.

Broadcast Receivers. Broadcast Receivers allow apps to receive events from the system and to answer such system-wide broadcast announcements. The system can also deliver broadcasts to apps that are not running due to the fact these events are outside the standard user data flow. Broadcast receivers, activities, and services are activated by a message called an 'intent'. Intents define which actions activities and services should perform, and what announcements broadcast receivers should be shared.

Content Providers. Content Providers are able to read and write shared app data that is stored in any location where an app has access to it. If allowed by the content provider, other apps can query and modify the data. Content providers are not activated by intents, rather they are activated from targeted requests from a content resolver.

All of these app components must be declared in an app's 'Manifest' file. Here the Manifest, normally found as AndroidManifest.xml, declares the app components but also identifies permissions, the minimum API level necessary, API libraries, and software and hardware components needed by the app [3]. Additionally, Android apps have a number of resource files that are separate from app code. These files are found commonly in the /res folder found in the APK file. Most of these resources deal with the layout of an app, fonts, styles, or strings, etc. [4].

2.3.3 App Development Overview

We illustrate the Android development steps in Figure 2.2. While anyone can build and develop an Android app, in order for the app to be published, one must register a developer account with Google Play. Then, developers can publish their app on Google Play which goes through a security vetting process. Once the app passes the review process, it is available for download on Google Play. The review process may take from 2-7 days according to Google [87]. There have been significant improvements and more stringent vetting over the years as Google Play has developed, yet there are still lapses that occur allowing for potentially malicious apps to slip through and masquerade as a legitimate app. We will discuss this further in Section 2.4.

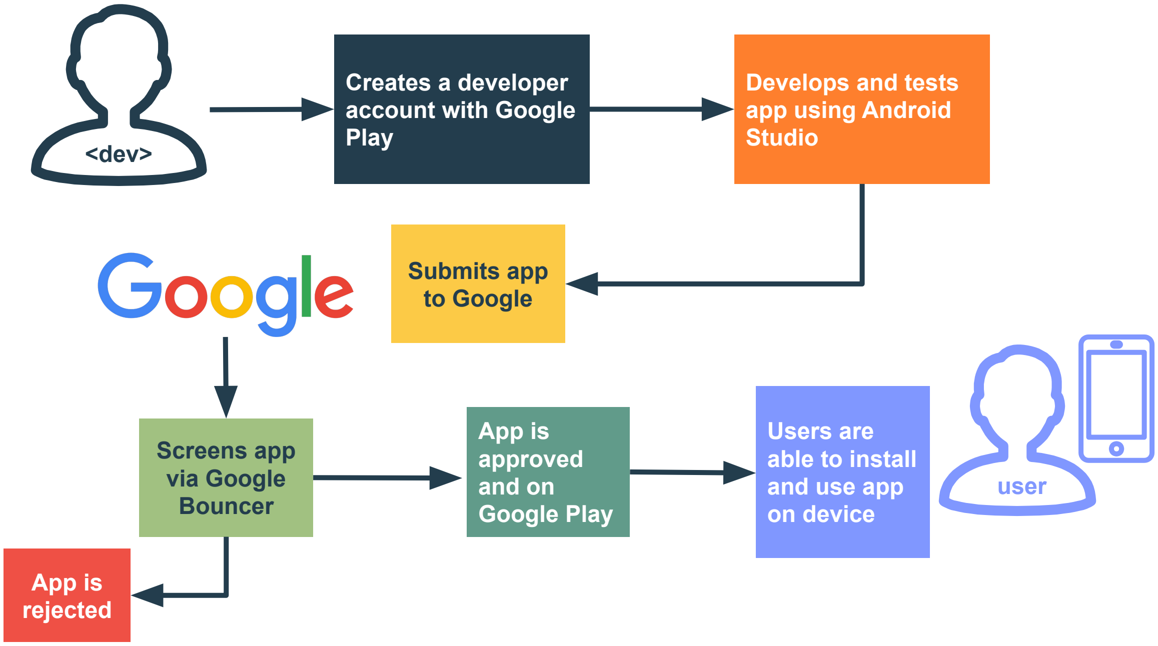

<dev>
Creates a developer account with Google Play
Develops and tests app using Android Studio
Submits app to Google
Google
Screens app via Google Bouncer
App is approved and on Google Play
Users are able to install and use app on device
user
App is rejected

Figure 2.2 Android development overview.

2.4 Mobile App Security

Mobile apps are lucrative targets for attacks due to their large user base. Just like traditional desktop apps, mobile apps can be exploited to extract user data. As a result, mobile security is a necessary subject of study to better understand platform and developer practices as well as methods on how apps can be exploitative and malicious.

2.4.1 Permissions

The Android system applies the ‘principle of least privilege’, meaning that apps have access to components required to run and nothing more [6]. Apps must be granted permission by users in order to to have access to such capabilities. Permissions give apps access to data and restricted actions. Certain permissions provide access to sensitive user data, making these permissions more critical than others. For example, apps having access to Call Log or SMS permission groups must have a core functionality related to the permissions [12]. Other sensitive permissions include

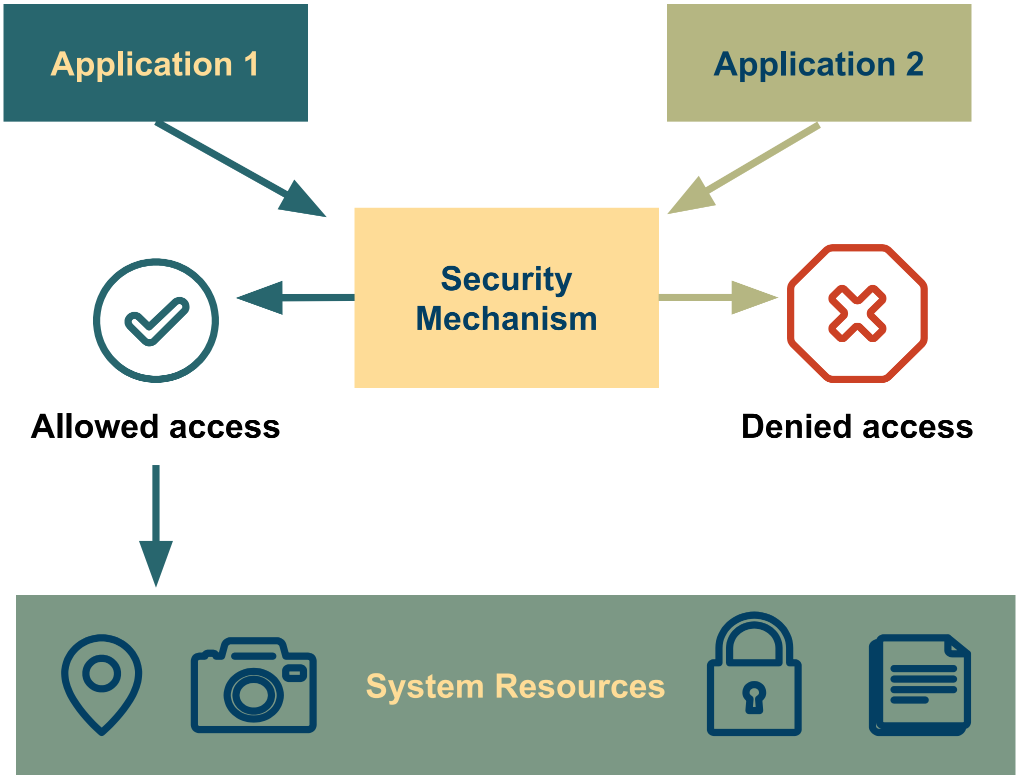


Figure 2.3 Android security model.

those related to location data or manipulating external storage and files. There are different types of permissions that can be granted to an app depending on its purpose and data necessary to function.

- Install time permissions: granted by system upon app installation.
- Runtime permissions: have access to restricted data and actions but require users to grant access first.
- Special permissions: defined only by OEM or platform.

Figure 2.3 illustrates how permissions provide security in Android.

Permission Abuse. Despite requiring user consent to have access to critical functions, permissions are easily abused and expose users to malicious behavior. Many times, developers do not realize that they are requesting unnecessary permissions which result in potential privacy threats and risks. Alenezi et al. have found that over 80% of apps request more permissions than what is actually needed [32]. Google has tried to mitigate permission abuse by implementing policies towards developers to curb them from using needless permissions in their apps. Developers would receive a warning from Google regarding the amount of permissions found and roughly 60% of developers complied and removed the excess permissions [130]. Since users provide access to functions via permissions, malicious apps take advantage of this and are able to use critical functions, hijacking a user's device. For example, the vulnerability known as "Strandhogg" exploited the permission prompt on Android, which ironically is used as a security measure – asking the user first to use those capabilities. Essentially, malicious apps display a fake permission pop up once the user attempts to open a legitimate app. Unknowing users assume that they are granting permissions to a trusted app without realizing it is malware [86]. In later chapters, we will discuss other examples of permissions abuse.

2.4.2 Android Fragmentation

Fragmentation refers to the vast variety of Android versions that are currently being used on consumer devices [19]. Since Android is an open source OS, fragmentation across versions and devices is an issue that is difficult to solve. Third party manufacturers and carriers are inconsistent with updates on their devices causing a lagged distribution of the latest version to users. This results in a severe lapse in security, as Google releases new security updates monthly. Thus, a user who has an outdated version of Android can be subject to a litany of exploits. For instance, a remote code execution attack occurred in 2012 on the JavaScript Interface in Android [139]. The JavaScript Interface in Android is a feature which allows apps to expose app-level objects to JavaScript code in a WebView. Because of this, malicious apps could could access Java APIs by simply calling calling `getClass()`. Google addressed this vulnerability in API levels 17 and higher; however, the change is not reflected in API levels 16 or lower, meaning that apps with that API level can still be exploited

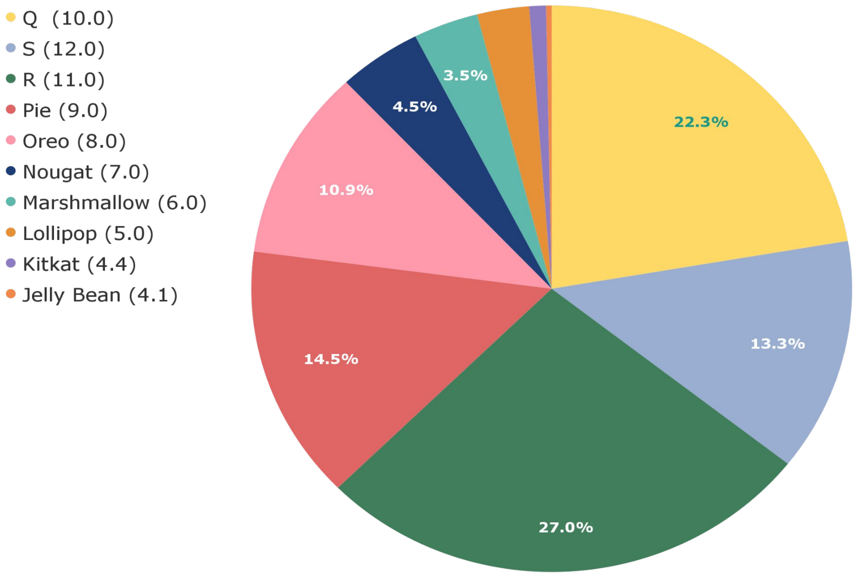


Figure 2.4 Android API distribution as of August 2022 [2].

[105]. Nevertheless, Google is trying to bridge the gap with newer versions' adoption rate. This is evident with the current API distribution, with recent versions, such as Android 12 (or S) trying to keep up with Android 11 (R) as of August 2022 [2]. We can see the overall API distribution in Figure 2.4. Still, having only roughly 40% of devices running the most recent versions is still very concerning and a potential security risk.

2.4.3 Malware

Mobile vulnerabilities have existed prior to Android. In 2004, the first mobile virus, Caribe, targeted Nokia phones. While it was not inherently damaging towards the user, Caribe did use Bluetooth to transfer files to other Nokia phones and would cause excessive battery drain. In 2005, with the Trojan Skulls, users had to manually install the malware on their Series 60 device. The malware would render the device unusable by overriding the system app by creating new files with the same name in the folder.

In 2010, another malware, known as FakePlayer, was found in the Android ecosystem. While FakePlayer had to be manually installed, like most other malware, it directly affected the user involved rather than just the device. FakePlayer would send premium SMS messages, costing $5 each, while running in the background [61]. Soon this would be the pattern for more malicious apps that would attempt to steal user data. A study in 2012 shows that one of the most popular types of malware found on Android were premium service abusers, such as FakePlayer [114].

Zhou et al. [161] provide a comprehensive characterization of existing Android malware and has generalized methods of how they end up on a user's device. Malicious apps may repackage popular apps by adding dangerous payloads and resubmit them in app marketplaces, causing users to inadvertently download the infected version of the app. Another method involves adding a malicious payload as an update component to fetch during runtime. Some forms of malware attempt to entice users into downloading their app and have users click in-app advertisements which redirect them to a malicious website where they can download malicious apps. Additionally there are fake apps which masquerade as legitimate apps, such as FakeNetflix, which steals account credentials.

In order to keep up with the changing platform, malware constantly evolves to evade detection and in functionality. One example is Anserverbot, which employs dynamic loading to prevent itself from being revealed by anti-virus software. Additionally, the malware also has the capability to detect installed anti-virus software. Upon discovering anti-virus software, Anserverbot is able to stop the app from running [161]. As discussed, older viruses and malicious programs on mobile devices were device oriented rather than focusing on users as the primary victim. Newer malicious apps target primarily user account information, messages, payment information, and other sensitive user data. An example of malware targeting critical user data is Mailbot. Found in June 2022, Mailbot steals banking details, cryptocurrency wallets, passwords, and hijacks SMS abilities. By taking advantage of SMS capabilities, Mailbot sends malicious messages to other users, thus distributing itself in the process. Additionally, Mailbot manages to bypass two factor authentication by being able to take screen captures of infected devices [107]. Since mobile devices contain vast amounts of personal data, malware threats will continue to exist, albeit in various forms.

2.4.4 Mobile Platform Policies and Enforcement

In the early stages of Android, there were few policies protecting users from downloading malicious apps from the Google Play Store. Android allowed at the time virtually any app to be on its marketplace. In 2012, Google introduced "Google Bouncer" to scan uploaded apps for suspicious behavior or evidence of malware [31]. A few months later with the release of Android 4.2 the "Verify Apps" feature was also implemented to scan all apps on a device for malicious behavior [117], which later became a feature found in a menu in Settings. In 2020, Google launched the Android Partner Vulnerability Initiative to manage security issues that are found with Android OEMs [20]. Despite these attempts to mitigate OS vulnerabilities, malicious apps still manage to slip through these checks. A recent study from 2020 shows that the Google Play store was the main distribution vector of Android malware citing almost 67% of malicious app installs originating from the Play Store [55]. Malware developers employ methods to mask functionality such as dynamically loading code until downloaded and executed, or uploading clean installations of SDKs to update it over time with malicious functionality [111]. This simply proves that there is more for Google to consider regarding policy enforcement and user protection.

2.5 Conclusions

We have seen how mobile devices and operating systems have evolved over the years. We focused on Android, one of the most popular mobile OS in the world, due to the vast number of users and its open nature. We discussed its structure and components as well as the development process in building an Android app. Additionally, we delved into mobile security, detailing vulnerabilities found in past mobile OSes as well as issues that still affect Android and Google's attempts to mitigate them. Now as we have introduced the appropriate background regarding the Android ecosystem, we can further discuss other points of research completed and planned in the following chapters.

CHAPTER 3

MOBILE AUTHENTICATION

Before we delve into mobile app security and deceptive behavior, we recall that mobile security begins from the moment a user attempts to unlock their device. As mobile devices have changed over the years, the methods of authentication have become more varied and developed in tandem. This chapter presents a user study on LG's 'Knock Codes' and quantifies the usability, as well as security, of this mobile authentication scheme.

3.1 Introduction

Designing new device unlock methods arise from a need of more security but also to provide the user with convenience. Implementing security without sacrificing usability and vice versa is a very fine balance to maintain; thus there are many known knowledge based authentication methods, such as Android graphical unlock patterns, PINs, and passwords. Due to the variations of mobile device unlock methods, there have been extensive user-based studies on the security of knowledge-based mobile authentication, including Android graphical unlock patterns [143, 40], PINs [46, 149, 103], as well as using passwords on mobile devices [104]. To add, LG developed Knock Codes as a new mobile authentication scheme designed to combat some of the vulnerabilities found in other unlock methods [137] and provide, per LG's advertising [138], "perfect security." Knock Codes require a user to recall a pre-selected series of at least 6 and at most 10 knocks[1] (or taps) on a 2×2 quadrant which is displayed upon setup and can be entered with the phone screen on or off. Knock Codes are used less frequently than PINs or Android patterns, but we estimate

[1] In earlier models, like the 2014 LG G2 [140], where this method first appeared, codes required at least 3 and at most 8. Newer models require 6 to 10 knocks occurring in at least 3 quadrants.

that there is a large number of Knock Code users, 700,000–2,500,000 in the US alone. Since Knock Codes have not been studied, we performed a joint study[2] to evaluate the security and usability of Knock Codes. We conducted two online user studies on Amazon Mechanical Turk: a preliminary study ($n = 218$) and a main study ($n = 351$), analyzing a total of 1,138 Knock Codes (436 in the preliminary study and 702 in the main study). In the main study, we evaluated three between-group treatments: a control treatment, where participants used the current 2x2 Knock Code interface; a blocklist treatment, where participants selected 2x2 Knock Codes with some popular codes, as measured in the preliminary study, being disallowed; and finally, a big grid treatment, where participants selected Knock Codes on a larger, 2x3 grid. While seeming like a straightforward attempt to increase security, an expanded Knock Code grid to 2x3 does not increase, and sometimes worsens, security compared to 2x2 Knock Codes. After 30 attempts, a simulated attacker correctly guesses *more* 2x3 Knock Codes compared to 2x2 (41 % vs. 37 %). However, blocklisting common Knock Codes (as collected in the preliminary study) is more effective at improving guessing security: only 19 % of these codes were guessed within 30 attempts in simulation. Overall, participants perceived Knock Codes (across treatments) as secure; however, among all treatments, participants were more hesitant to rate Knock Codes as *more secure* than PINs, Android Unlock Patterns, or alphanumeric passwords. Despite the fact that participants reported Knock Codes as “simple” and “memorable”, responses to the SUS [48] questions averaged to “marginal” or “ok” usability (69.8, 68.1, and 64.3, for the control 2x2 treatment, the larger 2x3 treatment, and the blocklist informed 2x2 treatment, respectively). Entry and recall times for Knock Codes were also much slower than what was reported for PINs and Android patterns [76, 103], suggesting lower usability.

[2] Joint work appeared as: *Knock, Knock. Who’s There? On the Security of LG’s Knock Codes* by Raina Samuel, Philipp Markert, Adam J. Aviv, and Iulian Neamtiu at USENIX Symposium on Usable Privacy and Security (SOUPS) 2020 Virtual Conference.

These results indicate that users are interested in new forms of mobile authentication, in particular ones that have options for unlocking with the display off. However, given the usability and security challenges of Knock Codes, *we would not recommend further deployment as currently configured.* For users and developers who wish to continue to use Knock Codes, we would recommend using a blocklist to inform selection as it provides increased security with small effects on usability.

3.2 Methodology

We collected data via Amazon Mechanical Turk (MTurk) using an online survey whereby participants were directed to use their mobile devices (checked via the user-agent) to select *two* Knock Codes as well as answer general questions about Knock Codes and their demographics. The two Knock Codes were primed based on different security scenarios, as informed by prior work of Loge et al. [98]. We found some, but minor, differences between Knock Codes in each scenario, similar to Loge et al.'s findings for Android patterns.

We conducted two studies: a preliminary study and a main study which is based on the preliminary study and presented here. The main difference between the two studies is that the main study was focused on participants using mobile devices while the preliminary allowed participants to use traditional computers. From the preliminary study, we were able to refine the main study as well as develop a blocklist of the 30 most common Knock Codes selected in the preliminary study (see Table 3.1).

Estimating US Knock Code Usage. We generalized our participants' device usage and authentication methods based on age, normalizing the demographic to the US population using census data [144, 145]. We saw that LG's market share in the US had a range between 8% to 12% among the estimated 285,300,000 smartphone users [56, 133]. Using that, as well as a 95% confidence interval, as our lower and upper

Table 3.1 Top 28 4-Grams with at Least 13 Occurrences

Rank	4-Gram	No.	%
1	▘ ▝ ▖ ▗	68	9.1 %
2	▘ ▝ ▗ ▖	62	8.3 %
3	▗ ▖ ▘ ▝	47	6.3 %
	▝ ▗ ▖ ▘	47	6.3 %
5	▘ ▘ ▝ ▝	41	5.5 %
6	▖ ▗ ▘ ▝	37	4.9 %
	▝ ▖ ▗ ▘	37	4.9 %
8	▘ ▗ ▝ ▖	31	4.1 %
9	▗ ▝ ▖ ▘	27	3.6 %
10	▝ ▖ ▘ ▗	26	3.5 %
	▘ ▘ ▗ ▗	26	3.5 %
12	▘ ▝ ▝ ▗	24	3.2 %
13	▗ ▖ ▝ ▘	23	3.1 %
	▘ ▗ ▖ ▝	23	3.1 %
15	▖ ▘ ▝ ▗	20	2.7 %
16	▖ ▝ ▗ ▘	19	2.5 %
	▝ ▝ ▖ ▖	19	2.5 %
	▘ ▖ ▗ ▝	19	2.5 %
19	▘ ▖ ▝ ▗	18	2.4 %
20	▝ ▝ ▗ ▗	17	2.3 %
21	▗ ▗ ▖ ▖	16	2.1 %
	▖ ▗ ▝ ▘	16	2.1 %
	▝ ▘ ▖ ▗	16	2.1 %
24	▝ ▖ ▗ ▗	15	2.0 %
25	▝ ▗ ▘ ▖	14	1.9 %
	▘ ▗ ▗ ▝	14	1.9 %
	▘ ▝ ▝ ▖	14	1.9 %
28	▖ ▝ ▘ ▗	13	1.8 %

bounds, we conclude that there are potentially many Knock Code users: 728,693 to 2,567,207 in the US alone. We believe, though, that the actual adoption rate is most likely on the lower end. While an optimistic estimate, the numbers still suggest that there is a substantial number of Knock Code users in the general public, particularly worldwide. Even though Knock Codes are not as widely adopted as other traditional methods of mobile authentication, it is still important to study user behavior with real-world, deployed authentication systems. In addition, on Google Play many Knock Code apps can be installed on any Android device, thus not limiting

Table 3.2 Top 30 Most Frequent Knock Codes in Control 2x2

Rank	Knock Code	No.	%
1	▘ ▝ ▖ ▗ ▘ ▝	16	6.9 %
2	▘ ▝ ▗ ▖ ▘ ▝	9	3.9 %
3	▘ ▘ ▝ ▝ ▖ ▖	8	3.5 %
4	▘ ▖ ▝ ▗ ▘ ▖	6	2.6 %
5	▘ ▗ ▝ ▖ ▘ ▗	5	2.2 %
	▘ ▘ ▗ ▗ ▝ ▖	5	2.2 %
6	▗ ▘ ▖ ▝ ▗ ▘	4	1.7 %
	▖ ▘ ▝ ▝ ▘ ▖	4	1.7 %
	▘ ▖ ▗ ▝ ▘ ▖	4	1.7 %
	▘ ▝ ▗ ▘ ▝ ▗	4	1.7 %
	▘ ▝ ▖ ▗ ▘ ▝ ▖ ▗	4	1.7 %
	▘ ▘ ▖ ▖ ▗ ▗	4	1.7 %
13	▝ ▗ ▖ ▘ ▝ ▗ ▖ ▘	3	1.3 %
	▝ ▗ ▘ ▖ ▝ ▗	3	1.3 %
	▝ ▝ ▗ ▗ ▖ ▖	3	1.3 %
	▘ ▗ ▖ ▝ ▘ ▗ ▖ ▝	3	1.3 %
	▘ ▗ ▖ ▝ ▘ ▗	3	1.3 %
	▘ ▗ ▝ ▖ ▘ ▗ ▝ ▖	3	1.3 %
	▘ ▝ ▖ ▗ ▘ ▗	3	1.3 %
	▘ ▝ ▖ ▘ ▝ ▖	3	1.3 %
	▘ ▘ ▗ ▗ ▝ ▝	3	1.3 %
	▘ ▘ ▖ ▖ ▝ ▝	3	1.3 %
23	▗ ▖ ▗ ▖ ▘ ▝	2	0.9 %
	▖ ▘ ▖ ▘ ▖ ▘ ▝	2	0.9 %
	▝ ▖ ▗ ▘ ▝ ▖	2	0.9 %
	▝ ▖ ▘ ▗ ▝ ▖	2	0.9 %
	▝ ▘ ▖ ▗ ▝ ▘	2	0.9 %
	▘ ▗ ▖ ▝ ▗ ▖	2	0.9 %
	▘ ▗ ▖ ▝ ▘ ▘	2	0.9 %
	▘ ▗ ▝ ▖ ▖ ▗	2	0.9 %

Knock Codes to solely LG devices. For instance, the most highly rated Knock Code app on Android, “Knock Lock,” boasts more than 1 million installations and claims that it is an innovative lock screen that “will leave intruders baffled” [90]. This app is just one among the plethora of Knock Code knock-off apps that can be found on Google Play, indicating that this authentication method may have a higher adoption rate and influence on mobile authentication systems than appears initially.

Table 3.3 Top 30 Most Frequent Knock Codes in Blocklist 2x2

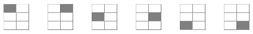
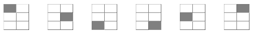
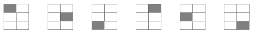
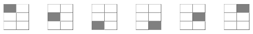
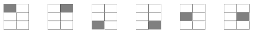
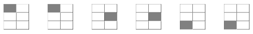
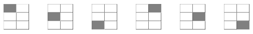
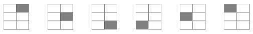
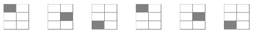
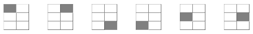
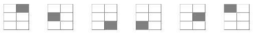
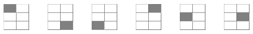
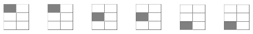
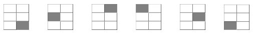
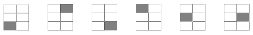
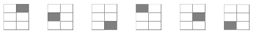
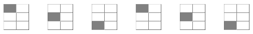
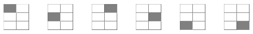
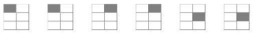
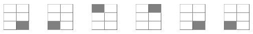
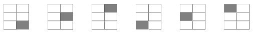
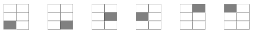
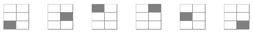
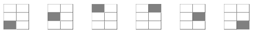
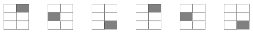
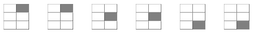
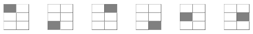
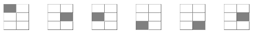
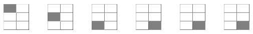
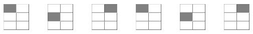
Rank	Knock Code	No.	%
1	▘ ▘ ▖ ▖ ▝ ▝	6	2.6 %
2	▝ ▝ ▖ ▖ ▘ ▗	5	2.2 %
	▘ ▘ ▗ ▗ ▝ ▝ ▖ ▖	5	2.2 %
4	▗ ▖ ▘ ▝ ▗ ▖	3	1.3 %
	▗ ▝ ▘ ▖ ▗ ▝	3	1.3 %
	▖ ▘ ▝ ▖ ▘ ▝	3	1.3 %
	▝ ▖ ▘ ▗ ▝ ▖	3	1.3 %
	▘ ▖ ▗ ▗ ▖ ▘	3	1.3 %
	▘ ▖ ▗ ▝ ▘ ▖	3	1.3 %
	▘ ▖ ▝ ▗ ▘ ▖ ▝ ▗	3	1.3 %
	▘ ▝ ▗ ▖ ▗ ▝	3	1.3 %
	▘ ▝ ▗ ▖ ▘ ▝ ▗	3	1.3 %
	▘ ▝ ▖ ▗ ▘ ▝ ▖ ▗ ▘ ▝	3	1.3 %
	▘ ▘ ▗ ▗ ▖ ▝	3	1.3 %
	▘ ▘ ▝ ▝ ▖ ▗	3	1.3 %
16	▗ ▗ ▖ ▖ ▝ ▘	2	0.9 %
	▗ ▗ ▖ ▖ ▘ ▘ ▝ ▝	2	0.9 %
	▗ ▘ ▖ ▝ ▗ ▘	2	0.9 %
	▖ ▘ ▝ ▗ ▖ ▘	2	0.9 %
	▝ ▖ ▗ ▘ ▝ ▖	2	0.9 %
	▝ ▝ ▖ ▖ ▗ ▗	2	0.9 %
	▝ ▝ ▘ ▘ ▗ ▗ ▖ ▖	2	0.9 %
	▝ ▝ ▘ ▘ ▖ ▖	2	0.9 %
	▘ ▗ ▝ ▖ ▘ ▗ ▝ ▖	2	0.9 %
	▘ ▖ ▝ ▗ ▗ ▘	2	0.9 %
	▘ ▖ ▝ ▘ ▖ ▝	2	0.9 %
	▘ ▝ ▗ ▖ ▘ ▗	2	0.9 %
	▘ ▝ ▖ ▗ ▝ ▘	2	0.9 %
	▘ ▝ ▖ ▖ ▗ ▘	2	0.9 %
	▘ ▘ ▗ ▖ ▝ ▘	2	0.9 %

3.3 Results

The first step in analyzing Knock Codes is to determine the frequency statistics. Table 3.2, 3.3, and 3.4 display the 30 most frequent patterns across the scenarios for three treatments of the main study. The frequencies which we observed in the preliminary study are shown in Table 3.1. The preliminary study codes and the con-2x2 codes have a lot of overlap, with 42.0% of the Knock Codes from the preliminary study appearing in the top-30 most frequent codes in the Control 2x2 treatment. This helps justify using the most frequent preliminary study codes as the basis of the blocklist for the bl-2x2 treatment.

The frequencies of the Knock Codes show different characteristics depending on the assigned treatment, suggesting natural, human tendencies in the selection that

Table 3.4 Top 30 Most Frequent Knock Codes in Large 2x3

Rank	Knock Code	No.	%
1		11	4.6 %
2		10	4.2 %
3		9	3.8 %
		9	3.8 %
5		8	3.4 %
6		7	2.9 %
		6	2.5 %
8		5	2.1 %
		5	2.1 %
		5	2.1 %
11		4	1.7 %
		4	1.7 %
		4	1.7 %
14		3	1.3 %
		3	1.3 %
		3	1.3 %
		3	1.3 %
		3	1.3 %
		3	1.3 %
		3	1.3 %
21		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %
		2	0.8 %

can be leveraged in predicting and guessing Knock Codes. We take advantage of this observation when guessing codes. Participants in the blocklist group use more repeated taps whereas codes created for the 2x3 treatment make use of the larger grid and follow directional patterns. Knock Codes created for the control depict a mix and follow both strategies equally.

Start/End Quadrant Frequency. The foremost property that can be leveraged is the tendency to start and end in certain quadrants. There is a strong tendency to begin codes in the upper-left. Similar observations were made for Android Graphical

Patterns [143] and is likely due to the left-to-right nature of the English language which is dominant among our participants. The least common starting points in the preliminary study as well as the control and blocklist treatment were in the lower row. In the big treatment, on the other hand, the middle row is used the least often. We found significant differences between both the control and big treatment ($p < 0.001$) as well as blocklist and big ($p < 0.001$), suggesting that the larger grid size affected how participants chose to start and end their codes.

Code Length. We also analyzed the Knock Codes with respect to length. The average code length was 6.4, 6.5, and 6.2 in each treatment, con-2x2, bl-2x2, and big-2x3, respectively. We observed statistical differences using ANOVA ($f = 11.57, p < 0.001$) between the treatments. In post hoc analysis, using pairwise t-test comparison, the difference lies primarily in the longer big-2x3 Knock Codes, which was statistically different from both bl-2x2 ($p < 0.001$) and the con-2x2 ($p < 0.001$). Surprisingly, the larger grid size encouraged slightly shorter Knock Codes. Regardless, the vast majority of Knock Codes were of length 6, which was the median value, or 8, with a few codes of length 10.

Security Analysis. Across all comparisons, we find that Knock Codes in the control 2x2 are significantly weaker in terms of their guessability. This means Knock Codes as they are currently deployed are more guessable than both 4- and 6-digit PINs as well as Android Patterns. When considering the First-Entry dataset, the differences are less distinct, but even in this ideal case, the inferiority of Knock Codes remains. Aviv et al. conjectured, and we do so here as well, that there may be a false sense of security that the larger set of choices offers, whereby users believe their individual choice matters less in the face of the increased number of possibilities. Analyzing other grid sizes, such as 3x2 or 3x3, would offer additional insight; nevertheless, providing more complexity in how to select Knock Codes does not increase the security. Finally, we observed strong security improvements with the introduction of a blocklist. As compared to the control, the blocklist cuts the success rate of an attacker within the first 30 attempts by 30 % to 50 % and increases the guesswork by roughly 1.5 bits when considering an offline attacker. The blocklist clearly encouraged more diverse choices but also had the side effects of increasing user frustration and usability, as we describe later.

Usability Analysis. In this section, we focus on the usability metrics of Knock Codes. We first report results on the setup and recall times. Afterwards, we will focus on memorability and recall rates within our study, followed by the qualitative and quantitative responses to security and usability prompts.

Setup and Recall Times. Participants needed on average 16.2 s and 18.4 s to select and confirm a 2x2 and 2x3 Knock Code, respectively. This is faster than the blocklist treatment (22.5 s), where participants also had to make more attempts due to blocklisting (1.5 vs. 1.1 attempts). In comparison, setting up a 4- or 6-digit PIN takes on average only 7.9 and 11.5 seconds respectively [103] which is significantly faster than Knock Codes. While the described discrepancy between Knock Codes and PINs is distinct, the numbers for PINs may be lower since users are presumably more familiar with PINs as compared to Knock Codes. The differences may decrease with increased familiarity with Knock Codes. In terms of the recall, which can be compared to unlocking a smartphone, the 2x2 (7.2 s per attempt) and 2x3 Knock Codes (7.1 s per attempt) are more efficient than Knock Codes selected with a blocklist (7.4 s per attempt). With 1.2 attempts per entry, participants took 11.3 seconds to enter their Knock Codes for the blocklist treatment Compared to entering an Android pattern (3.0 s) or a PIN (4.7 s) [76], clear usability issues with Knock Codes emerge as entering them is twice as slow. With greater use of Knock Codes, these differences may decrease. However, we find it unlikely that Knock Codes will be as efficient to use as patterns or PINs.

Memorability. Memorability depicts an important benchmark for any authentication method. We analyzed the memorability of Knock Codes by looking at the recall rates at the end of the survey. While recall rates are an imperfect measure for the memorability, as the survey took most participants less than 10 minutes to complete, it does speak to potential underlying usability issues, particularly if codes were not properly recalled in this short window. We separated the recall rates based on each treatment. The con-2x2 treatment participants successfully recalled their codes 88.8 % of the time. The participants with the larger 2x3 grid had higher recall rates of 92.9 %, which may suggest an interesting usability vs. security trade-off as this group chose shorter and also some of the weakest Knock Codes. However, we did not find significant differences between the con-2x2 and big-2x3 recall rates using a χ^2 test. We would expect long term memorability rates to be equally high, but further studies would be needed to confirm that conjecture. The worst recall rate came from participants in the bl-2x2 treatment: 80.6 % successfully recalled their Knock Code, and the result was significantly different from the other two recall rates ($p < 0.0001$ for both comparison tests). This could be attributed to the impact of the blocklist, where participants who hit the blocklist had lower recall rates (66.0 %) than those that did not (84.9 %). Most likely, the blocklist affected users in two ways. First, participants who chose blocklisted codes were forced to select multiple codes until landing on one that was not blocklisted. The average number of blocklisting events per user who hit the blocklist was 1.4. Second, the final Knock Code chosen ended up being more complex (as evident in the prior section), and thus harder to recall. Again, this suggests a clear trade-off between usability and security. We also analyzed the number of attempts to successfully recall a Knock Code. We found no statistical differences across all treatments between the attempts made in recalling the first or second scenario Knock Code correctly. In the big-2x3 treatment and the con-2x2 treatment, participants took an average 1.1 attempts when recalling a Knock Code correctly, with 3 attempts as the maximum. For the bl-2x2 group, users took on average 1.2 attempts to correctly recall a Knock Code, again having a maximum of 3 attempts. Again, we find bl-2x2‘s result to be significantly different in terms of the number of attempts made in the other treatments ($p < 0.001$ vs. big-2x3 and $p < 0.001$ vs. con-2x2), thus showing that the blocklist has an impact on recalling Knock Codes, even for those participants that eventually correctly do so. It is important to note though, that users had a maximum limit of 3 attempts to recall their code before we considered it “cannot be recalled” for the purpose of expediting the survey. We also analyzed how participants failed to recall their Knock Codes by calculating the average edit distances between the submitted code and the true code for both recall attempts, one for each scenario. We determined that there was no statistical difference between the average edit distances among treatments. The average edit distance between correct and incorrect recalls was 3.6, suggesting that when users get a code wrong, they get it wrong by a large margin, as the median length Knock Code is 6.

User Responses. Users provided their opinions and insights regarding Knock Codes’ usability and security. We coded these free responses using two independent reviewers where disputes in coding were resolved until consensus was reached. The specific codes and their frequencies are presented in the Appendices. Overall Knock Codes were perceived positively by users, citing that they were “Easy,” “Quick,” and “Hard to Guess.” The uniqueness of Knock Codes also appealed to users who indicated they especially liked the fact that it is a “Discreet” and “Secure” authentication method which can be inconspicuous and hidden from others. For many of the participants, Knock Codes were a new method of authentication. Users employed various tactics when choosing their Knock Codes. We observed such strategies in determining memorable yet secure codes. To make the Knock Code more memorable, the majority of users opted to use some sort of “Pattern” or “Variation” that would be “Simple.” Other techniques users employed include “Directional,” “Shape,” “Game,” and “Repeated.” Often, users would create codes based on something “Personal” to them, such as the letter of a word that had meaning to the user. While many users did not have a specific strategy for security and still focused on making their code “Easy to Remember” as the main priority, others determined that using “All Quadrants” or “ Multiple Regions,” as well as making the code “Long” or “Random” or being “Unexpected” and “Different” would secure their codes. Making their codes “Hard to Guess” often included attempts to obfuscate the number of clicks and the regions, using speed and potentially unpredictable tactics. Users continued to use similar tactics for memorability to double as security in their Knock Codes, for instance having “Repeated” regions. Upon comparing Knock Codes with other forms of security, on average, users found passwords, PINs, and Android patterns to be more secure than Knock Codes (see Figure 3.1). Overall, users found Knock Codes adequately secure, i.e., being difficult to hack, resistant to smudge attacks and shoulder surfing. However, they were not completely convinced about Knock Codes’ security. Users expressed what they disliked overall, specifically that they found Knock Codes “Hard to Remember” and “Insecure,” paving the way for an attacker to easily guess a Knock Code. They also found the interface provided “No improvement” and disliked how it was “ Hard to type-in” the Knock Codes.

To have a more general opinion of the overall usability of Knock Codes, we employed the System Usability Scale (SUS). The average response for the con-2x2 treatment is 69.8, the big-2x3 is 68.1, and the bl-2x2 is 64.3. These scores are generally

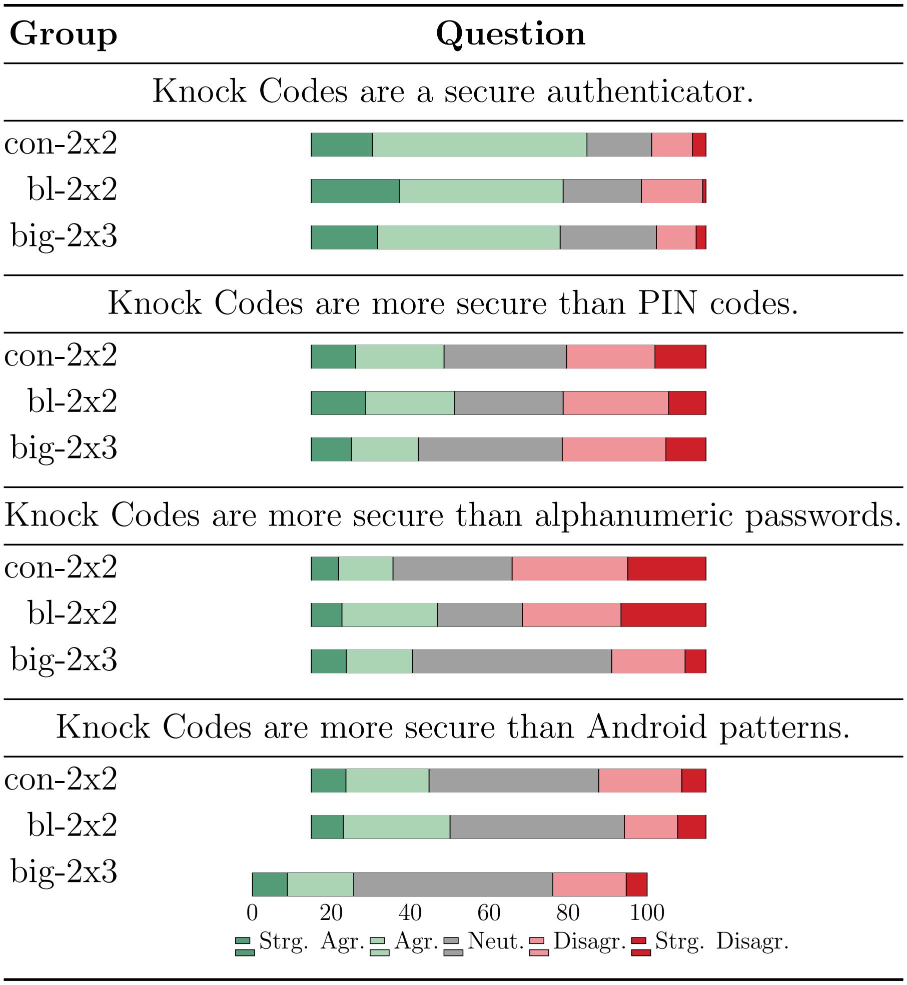


Figure 3.1 Likert response to comparisons to other mobile authentication methods.

rated as "ok" or "marginal," with only the control treatment potentially offering some above-average usability.

3.4 Conclusions

Overall, while most participants offered some positive thoughts, their perception of the security of Knock Codes lagged behind other deployed options, and the SUS values for all schemes were "ok" or "marginal." The positive feedback regarding Knock Codes suggests an openness to new designs in mobile authentication, particularly to authentication that can be entered while the phone screen is off. There were also increased perceptions of security from targeted attacks. It is reasonable to view Knock Codes as offering new design concepts that can ultimately improve mobile authentication. However, we find that Knock Codes, as currently deployed, provide weaker security than other available knowledge-based, mobile unlock methods, such as 4-/6-digit PINs and Android patterns. This is far from the "perfect security" promised by LG's advertisement of Knock Codes. As such, we cannot recommend deploying Knock Codes in their current form as compared to alternative authentication options. Our results also indicate that a straightforward improvement such as increasing the grid size to 2x3 may offer little or worse security. Blocklisting common Knock Codes, on the other hand, does provide more resilience to a throttled attacker, as has been found in password authentication [74] and PINs [103]. Yet, blocklisting runs the risk of increasing user frustration during selection, but since selecting a Knock Code is a one-time event, the usability trade-off of adding a blocklist may be *extremely worthwhile* if Knock Codes continue to be available to LG users. Nevertheless, these results simply show the importance of maintaining a clear balance of security and usability when designing mobile authentication schemes for daily use. Neither facet of system design should be compromised in order to bolster the other as such lapses result in unusable and insecure systems, in this case, authentication. Mobile authentication is one of the first layers of security for a user's device to be protected from any physical threats and is very important to not be overlooked in security research. As we have looked at the first aspect of mobile security, we will now look into forms of deceptive app behavior and system vulnerabilities once a user has access to their device.

CHAPTER 4

SELF HIDING BEHAVIOR IN ANDROID APPLICATIONS

We will now discuss deceptive app behavior found among apps in Android. Deceptive app behavior begins with apps that conceal certain activities from users. This chapter details and observes the use of 12 "self hiding" behaviors found in Android apps.

4.1 Introduction

Apps which hide deceptive behavior should be treated as suspicious for both end users and app marketplaces. Primarily, app marketplaces or platforms should protect users from potentially malicious or suspicious behavior in apps. As a result, concealed activities and furtive behaviors are not disclosed to users, ultimately putting them at risk.

Mobile security research has mostly focused on malware activation, malicious payloads, permission abuse, or sensitive data leaks. Little attention has been paid to deceptive mechanisms that are essential for the success of malware, i.e., how malware manages to get installed, and continues to stay operating on the phone without the users noticing anything suspicious. To do so, malware uses a range of "self hiding" (SH) techniques, e.g., hiding the app, hiding app resources, blocking calls, deleting call records, or blocking and deleting text messages. However, SH behavior is not limited to simply malicious apps. Surprisingly, extremely popular "benign" apps such as *Airbnb*, *Instagram*, *Viber*, *Yelp*, and *Waze* also employ certain SH techniques in the name of user convenience.

In this joint work[1], we focus on characterization and detection of such techniques, e.g., hiding the app or removing traces, which we call "self hiding" (SH)

[1] Joint work appeared as: *Self-Hiding Behavior in Android Apps: Detection and Characterization* by Zhiyong Shan, Iulian Neamtiu, and Raina Samuel at International Conference on Software Engineering (ICSE) 2018 in Gothenburg, Sweden.

behavior. SH behavior has not been studied, rather it has been reported on only as a byproduct of malware investigations. We address this gap via a study and suite of static analyses targeted at SH in Android apps.

We believe that SHB is fundamentally deceptive and that having tools that perform accurate and early detection of SHB is key. First, Google Play or other app marketplaces, should be able to detect SHB, so that the existence of SHB can be considered in the decision to publish an app. Even if an app with SHB is published on the marketplace, users should be forewarned about the SHB so they can decide whether to install the app on the phone or not.

4.2 Defining SHB

In this section, we provide a comprehensive description of SH behaviors. We define *SH* as a behavior meant to hide the app or its actions from being viewed (or heard!) *by the user*. Note that we exclude those behaviors meant to evade security mechanisms, e.g., anti-malware tools or access control mechanisms – which have been studied thoroughly and are outside the scope of this chapter. Our characterization is based on manual analysis of about 200 malicious apps and automated analysis of about 3,000 other malicious apps. We found 12 SH behaviors; few are even mentioned in the research community, let alone characterized thoroughly, and some, including "Hide icon" and "Hide activity", are not mentioned at all. Users could employ three main approaches for identifying the presence of malicious apps: inspecting app objects (icon, app, activity), analyzing remote communication (SMS, MMS, and phone calls) or checking system reminders (system dialogs, sound, system logs, notifications, recent apps list, etc). Malware typically attempts to hide itself from these three aspects. To set up the discussion, in Figure 4.1 we show the number of SHBs in sample sets of 1,000 malicious and benign apps, respectively.

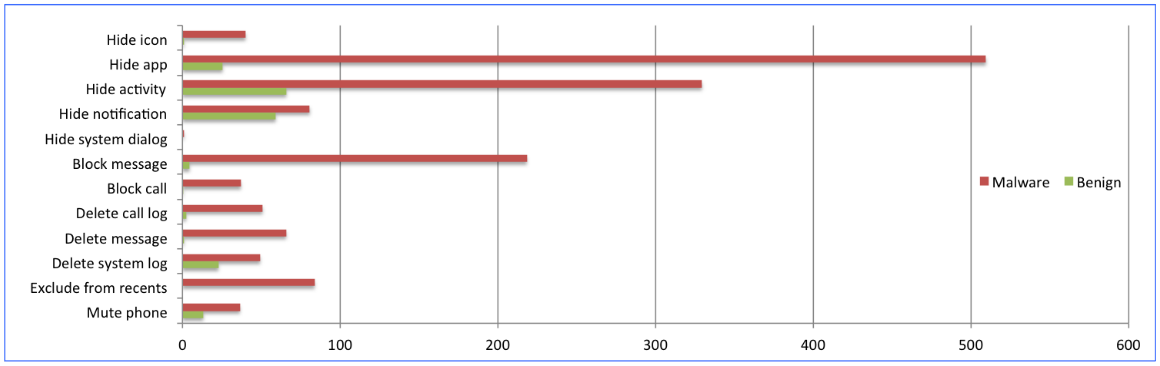


Figure 4.1 The numbers of SH behaviors in two sample sets of 1,000 malware apps and 1,000 benign apps, respectively.

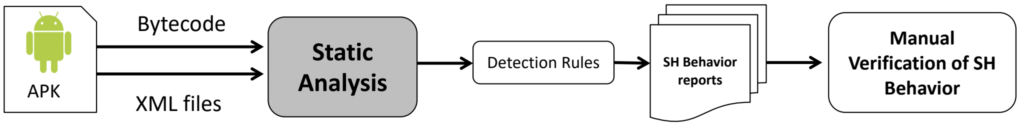


Figure 4.2 Tool overview.

4.3 Detecting SHB

We will now provide an overview of our tool; the tool design is shown in Figure 4.2. Using an APK file as input, we employed an automated static analysis[2] to produce an SH report, i.e., potential SH behavior. To verify the report, we performed a manual dynamic analysis of the behavior.

4.4 Self-hiding Behavior in Benign Apps

For each SHB category our tool has found in benign apps, we performed a targeted manual investigation – Why does this behavior occur? and What are the consequences for the user? – by focusing on widely-popular apps (e.g., with more than 100 million installs). This section summarizes some of our findings; we limit the discussion to 8 SHBs for brevity.

4.4.1 Hide App

Many popular benign apps, such as *Airbnb*, *Instagram*, and *Foursquare* start themselves as a service after receiving the BOOT COMPLETED event. This event, which requires the permission RECEIVE BOOT COMPLETED, notifies the app that the system has rebooted. Our tool reports this as "Hide app" SHB. Apps employ this technique as a means to initialize app-specific information and functions upon startup. While one could argue that the app is not hiding in the malicious sense (rather it is running in the background to have access to certain types of data – most commonly, location services), we believe that users should know when such apps are running: (1) so they understand why the battery is draining, and (2) so they understand the privacy implications of apps accessing and transmitting sensitive information (e.g., location) in the background.

4.4.2 Hide Notification

Well-known benign apps, such as *Truecaller*, *Viber*, and *Booking.com*use NotificationManager.cancel() or NotificationManager.cancelAll() to block notifications without user intervention. As a result these apps have been marked as having the "Hide notification" SHB. This is due to the nature of the command cancel() or cancelAll() which cancels all previously-shown notifications. Apps employ this technique as a means to update the user to the most recent notification or to consolidate notifications, especially in messaging apps such as Viber. Consolidated notifications may appear convenient to the user, however the app does not have a means to show high-priority notifications first (other than through chronological order). Therefore, users might prefer to receive notifications for all messages to reduce the risk of missing an important notification.

[2] The static analysis was not a contribution of this work.

4.4.3 Hide Activity

Our tool reports apps which use finish () and Window.addflags() or Window.setflags() as having the "Hide activity" SHB. Normally, apps use finish() to call the method onDestroy() as a means to kill the current activity and close any action the activity was managing, effectively bringing the user directly to the prior activity. We found that apps CleanMaster and BatteryDoctor use finish () for an activity called 'MarketCatalogActivity'. One can infer that this activity may check *what other apps are installed on the user's device*, possibly for advertisements, building a list, and killing the activity. After that, the apps may periodically poll the list to show ads for apps that users do not have. This behavior is intrusive to the user and can be easily manipulated to have activities which monitor user behavior in the background.

4.4.4 Mute Phone

Our tool discovered the use of AudioManager.setRingerMode() in the supposedly benign app *Camera360.* As its name states, this is a camera app which edits and takes photos, having more than 100 million installs and was "Best App of 2016 on Google Play in several countries".[3] The mere notion that a camera app has access to the device's ringer is suspicious. Our tool also discovered the "Mute phone" SHB in certain benign messaging apps like Viber and Whatsapp due to the use of Vibrator . cancel () and AudioManager.setRingerMode(). Even though it seems reasonable for messaging apps to exhibit this behavior, we believe that muting the phone should be done *by the user through a system-wide control* rather than silently by the app.

4.4.5 Block Message

As the BroadcastReceiver is usually a dormant app component, it is not surprising that the BroadcastReceiver's methods can be categorized as SH behaviors, especially

[3] https://play.google.com/store/apps/details?id=vStudio.Android.Camera360&hl=en

abortBroadcast(). As a result, many benign apps can exhibit this behavior. Interestingly, these apps are not limited to those which rely heavily on BroadcastReceiver. For example, the popular navigation app *Waze* uses abortBroadcast() which can be construed as the "Block Message" SH behavior. The abortBroadcast() method is used to prevent other receivers from obtaining the broadcast, thus blocking the communication. It might be justified that Waze employs this tactic as a means to prevent itself from getting location-based alerts that may be irrelevant or annoying to the user. While the intentions of the app appear benign, it removes decision-making from the user and can interfere with usability.

4.4.6 Block Call

Apps which use ITelephony.endCall() are considered to have the "Block Call" SHB. The benign app *Truecaller* has the sole purpose to identify and block spam calls, hence it was obviously marked to have this behavior. Despite explicitly stating that *Truecaller* automatically blocks calls, any app which decides for the user spam calls can be maliciously manipulated.

4.4.7 Hide Icon

'Hide icon' achieves its goal by manipulating category.LAUNCHER. While category.LAUNCHER merely indicates that activity should appear as an initial activity of a task, it is evident that apps can use it to promote other activities, masking itself beneath. Many popular benign apps such as Yelp, Accuweather, BBC, and ESPN have this behavior. By having various activities and controlling the launcher's top level apps, these apps should not have full autonomy to decide which activity is top-level.

4.4.8 Delete Call Log

Examples of this SHB include *Quick Heal Mobile Security*and *Camera 360*. Here the method used is fairly straightforward, via ContentResolver.Delete(). Again, while not typically dangerous, both of these apps are very suspicious to have such SH behavior. For instance, Quick Heal Mobile Security deletes the call log as part of the call filtering services it provides. However, the user does not know that this behavior is in fact a feature of this security application. On a similar note, Camera 360 as mentioned earlier, has nothing to do with a user's call logs in theory, yet the app shows signs of this behavior. Here, we feel that the marketplace hosting the app should scrutinize such dubious behavior further before publishing.

4.5 Conclusions

To conclude, we defined a set of self-hiding behaviors and verified static analysis reports to reveal such behavior. Our experiments indicate that the presence of self-hiding behavior is strongly associated with malicious behavior in a given app. Nevertheless, we also found plenty of benign, widely-popular apps that employ hiding techniques, which suggests that end-users and marketplaces would benefit from using an approach like ours to shed light on potential nefarious behavior in Android apps and improve overall user experience. This comes to show though that deceptive behavior is not limited to malware but can also be employed by trusted benign apps with the claim to aid the user. While we have revealed deceptive behavior that conceals activities from the user, this is only one facet of mobile security. In many cases, deceptive app behavior may occur from abusing capabilities or permissions that have been granted by the end user. In the following chapter, we will continue our study of deceptive app behavior regarding permission abuse.

CHAPTER 5

DEVICE ADMINISTRATOR ABUSE

As we have seen, apps are able to conceal behaviors and activities from users. However, deceptive behavior can also occur when providing an app access to certain sensitive capabilities. This chapter shows abuses regarding the device administrator functionality granted to apps. We found three types of device administrator abuses across various devices and Android versions.

5.1 Introduction

Device administration (DA) in Android provides capabilities in company settings to control over a wide range of security capabilities and features according to company policy, such as: enforce password strength and expiration; or lock/wipe devices remotely. Users, of course, do not realize that such critical permissions are granted to an app as they simply want the app to function as intended. These capabilities can be, and have been, abused. One such widely-abused capability is leveraging active DA permissions to prevent app uninstallation. To the best of our knowledge, there are no tools or studies for understanding the DA ecosystem: detecting DA use and abuse, characterizing benign and malicious DA behavior, understanding consequences of malicious behavior and recovery strategies, grouping malicious behavior into families, etc. In this joint work,[1] we fill this gap with an automated, effective and efficient approach to detect, and a study to characterize, DA-based abuse. Specifically, apps employ techniques to (1) conceal their DA status so the user is unaware of these apps' privileges, or (2) prevent the DA status from being deactivated – a practice that compromises device security and can render the device unusable until a factory

[1] Joint work appeared as: *Device Administrator Use and Abuse in Android: Detection and Characterization* by Zhiyong Shan, Raina Samuel, and Iulian Neamtiu at International Conference on Mobile Computing and Networking (MobiCom) 2019 in Los Cabos, Mexico.

reset. We name such apps *Deathless Device Administrator* (DDA) apps. We have studied a wide range of malware and benign apps that have DA permissions, and defined three DDA categories: DDA Reset, DDA Hide, and DDA Expert. We have discovered all three classes of DDA behavior in numerous apps that are still on Google Play.

5.2 Overview of Device Administrator

DA – a set of "extra" app privileges allowing tighter control, or even remote control, over Android devices – was originally introduced in Android 2.2 to facilitate enterprise apps and management of device fleets. Starting with Android 5, an alternative, superior set of features, "Android for Work", was introduced, but apps have continued to use, and unfortunately, abuse, DA. We first discuss the DA timeline and then the lifecycle of a DA app on a device.

5.2.1 DA Timeline

Android 2.2 (May 2010): DA policies introduced. DA support was introduced, supporting the following policies:

1. Enforcing password strength, reuse and expiration requirements, and forcing a password change.
2. Enforcing inactivity locks.
3. Forcing a certain storage area to be encrypted.
4. Disabling the camera.
5. Remote device locking.
6. Remote device wiping.

Android 5.0 (Nov. 2014) – Android 8.0 (Aug. 2017): Android for Work introduced, expanded. Android for Work (later, Android Enterprise, "AE" [37]) has introduced the concepts of managed devices, employed-owned devices, or work profiles; this is a set of features/policies somewhat similar to DA but broader, more secure, and with clearer roles.

Dec. 2017: Google announces planned DA deprecation. Google recommends that apps transition away from DA to AE and announces that DA will gradually be deprecated [141].

Android 9 (Aug. 2018): enterprise & soft deprecation. Starting with Android 9, DA was deprecated for enterprise use. For non-enterprise use, several policies (password expiration, disabling camera) were soft-deprecated, i.e., marked as deprecated but apps continue to function [36].

Android 10: hard deprecation. The aforementioned soft-deprecated policies were hard-deprecated starting with Android 10, i.e., apps targeting 2019+ API levels that attempt to use the policy will trigger a SecurityException. Nevertheless, at least three policies – forcing a device lock, wipe, password reset – will continue to be supported [36].

DA use and abuse. DA features can benefit organizations, IT admins, or parents. Unfortunately, when abused, DA can be turned against users. To uninstall a DA app, the user must first deactivate the app's DA capabilities, and then attempt to uninstall the app. Therefore, if the app can prevent deactivation, it can prevent uninstallation. For benign apps, the consequences can be a nuisance (unless the app is buggy, which can render the device unusable). For malicious apps, preventing uninstallation can mean unlimited/unfettered access and opens the door to abuse, e.g., ransomware. The "hard" DA deprecation in Android 9 (enterprise) and Android 10 (apps with 2019 target API) mitigates this issue. However in the remaining cases (non-enterprise apps, apps with API target <2019) the DDA potential persists: the underlying cause is the assumption that apps will cooperate when asked to give up DA privileges.

DA vs. Root. DA privileges do not require root access. "Rooting" a device (to gain root privileges) usually voids the device's warranty, whereas for an app to become DA the user simply needs to install the app and activate DA.

5.2.2 Deactivation Procedure

Next we will discuss how the DA status is deactivated by the user in normal conditions. By going to the Android *Settings* app, the user can deactivate the DA status for an app. When the user proceeds to cancel DA privileges for a DA app, the Settings app will invoke stopAppSwitch() to restrict activity switches for a period of time and then request the DA removal warning message from the app. Once the user confirms the action, DA privileges are deactivated by calling removeActiveAdmin(). Once the DA status is removed, only then, can a user uninstall the app. However, an app can interfere with this procedure to prevent users from deactivating the app's DA privileges, which ultimately prevents apps from being uninstalled.

5.3 Characterization

5.3.1 Benign Device Administrator Characterization

When installing a DA app, in theory the user should make an informed choice, and be familiar with DA app behavior or the policies the DA app enforces; in practice though, the user has close to zero knowledge of the consequences. Moreover, DA apps can come preinstalled. Therefore users may not understand entirely what these behaviors entail, or when it is appropriate to grant an app such privileges. In this section we discuss the most prevalent "benign" DA behaviors we found via a separate analysis, focused on benign DA usage, on a sample set[2] of 151 benign DA apps from Google Play. First we characterize the behaviors, then study which Google Play categories

[2] To ensure a representative sample of popular apps, we chose apps from across all 34 categories on Google Play; median number of installs across the sample set: 1,000,000+.

Table 5.1 Most Common DA Behaviors

Behavior	**# Apps**
Lock screen	68
Set password rules	34
Change the screen-unlock password	33
Monitor screen unlock attempts	32
Erase all data	31
Set lock screen password expiry	20
Disable cameras	19
Set storage encryption	16
Disable features in keyguard/screenlock	10

contain the highest concentrations of DA apps. The most common behaviors (by number of apps having that behavior) are provided in Table 5.1, and described next.

Lock Screen. This capability allows the DA app to control how and when the screen locks; it was most commonly found in screen lock apps, and also in parental control apps, antivirus apps, and enterprise management apps.

Set Password Rules. This controls the length and characters allowed in screen unlock passwords. There were 34 appearances of this behavior, in enterprise management apps, antivirus apps and remote phone security apps (which are used when a phone is lost or stolen).

Change Screen Lock. This behavior changes the screen lock password. There were 33 instances, primarily in enterprise and parental control apps. Unfortunately this capability can be detrimental if the implementation is buggy – the DA app can lock the device with a password or PIN unknown to the user, rendering the device useless until it is factory-reset.

Monitor Screen Unlock Attempts. This DA functionality monitors the number of incorrect passwords typed when unlocking the screen and will lock the phone or erase all the phone's data if too many incorrect passwords are typed. The functionality was used in 32 apps, mainly phone security apps (which is typical and the function of such apps), and enterprise management apps.

Erase Data. Phone data is erased without warning by performing a factory data reset; there were 31 cases, predominantly in security, antivirus, and enterprise management apps.

Disable Cameras. We found 19 instances, in enterprise management and security apps. As this is not a standard feature, disabling cameras might be puzzling or unsettling; based on the general nature of these apps, the user would expect the camera to work.

Storage Encryption. This encrypts the device's storage – a quasi-mandatory feature for enterprise management apps as well as phone security apps; we found 16 instances.

Disable features in keyguard/screenlock. This allows an app to disable the screen lock or any code that is involved with unlocking the device. Should it be misused, this is a potential major security breach for the device.

5.3.2 DA Across App Categories

As of March 2019, Google Play lists 34 app categories (aside from a 35th *Games* category which has its own subcategories). To find out which categories host the most DA apps, we performed a DA analysis on Top-600 apps in each category. Figure 5.1 shows the percentage of apps which fall into a certain category.

While *Tools* and *Business* are expected to be close to the top, surprisingly, *Parenting* had the second highest prevalence of DA apps. As to the reasons for requesting DA privileges, we note that 21% of the DA apps were in the *Tools* category with Lock Screen as the most prevalent DA behavior.

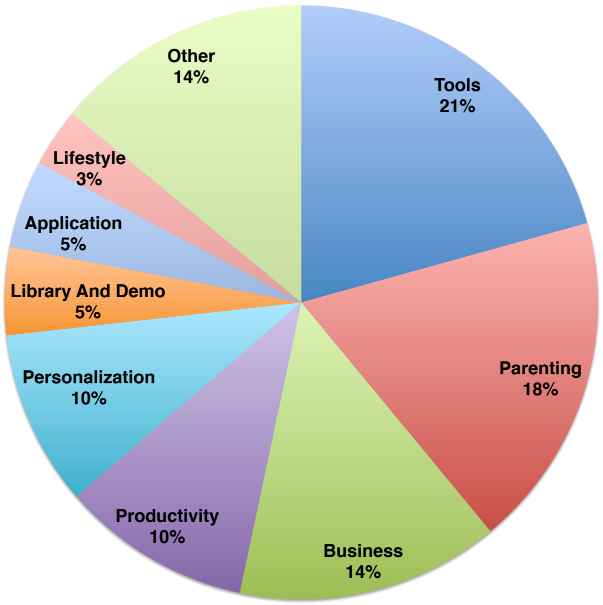


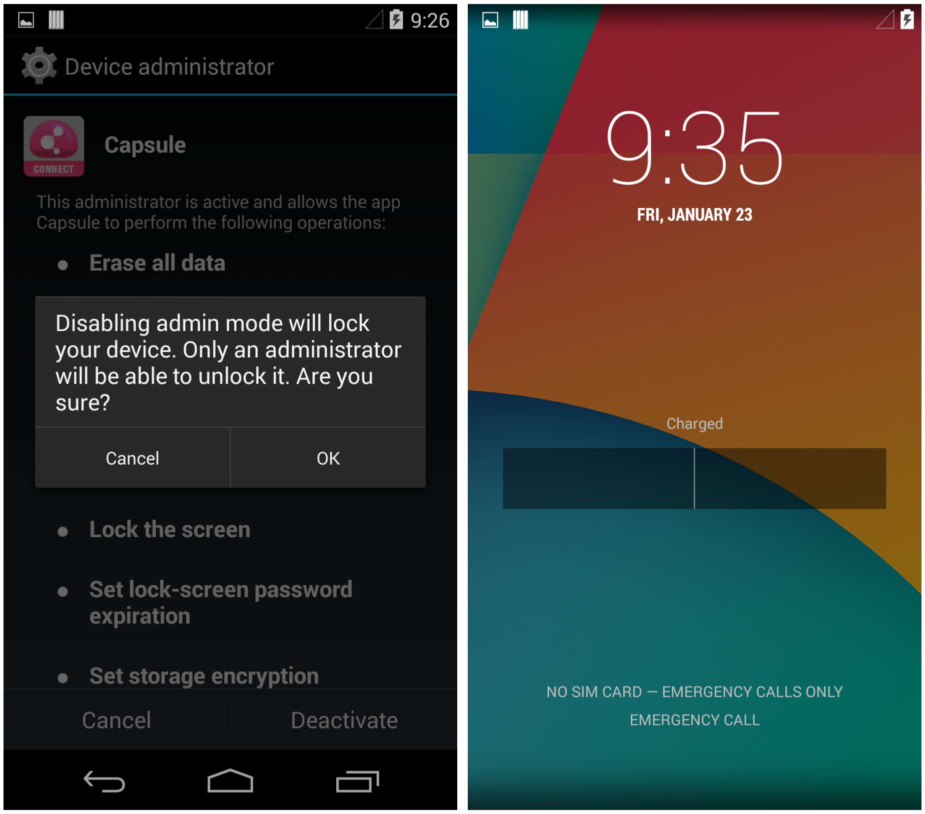
Figure 5.1 Percentage of DA apps per category.

Productivity and *Personalization* each make up 10% of the DA apps and they both have Lock Screen as their top DA behavior; in fact, for *Personalization* apps, Lock Screen is their only DA behavior. This is expected, as many *Personalization* apps are often various types of themed lock screen launchers.

5.3.3 Deathless Device Administrator

Deathless Device Administrator (DDA) apps represent DA apps that *prevent the user from uninstalling the app.* To do so, DDAs exploit vulnerabilities or weaknesses in the procedures Android uses for handling DAs. Accordingly, we introduce three types of DDA and exemplify that behavior on actual apps. We derived these three types of behavior via a semi-automated process (installing, checking DA status, attempting to deactivate DA) on 4,135 apps that had DA permissions. A static analyser tool was constructed to expose potential DDA behavior.

DDA Reset. DDA Reset apps prevent the user from disabling an app's DA capabilities; DDA Reset is irrecoverable – the only way to remove an app that uses DDA Reset is to restore the phone to factory settings.



This administrator is active and allows the app Capsule to perform the following operations:

- Erase all data

Disabling admin mode will lock your device. Only an administrator will be able to unlock it. Are you sure?

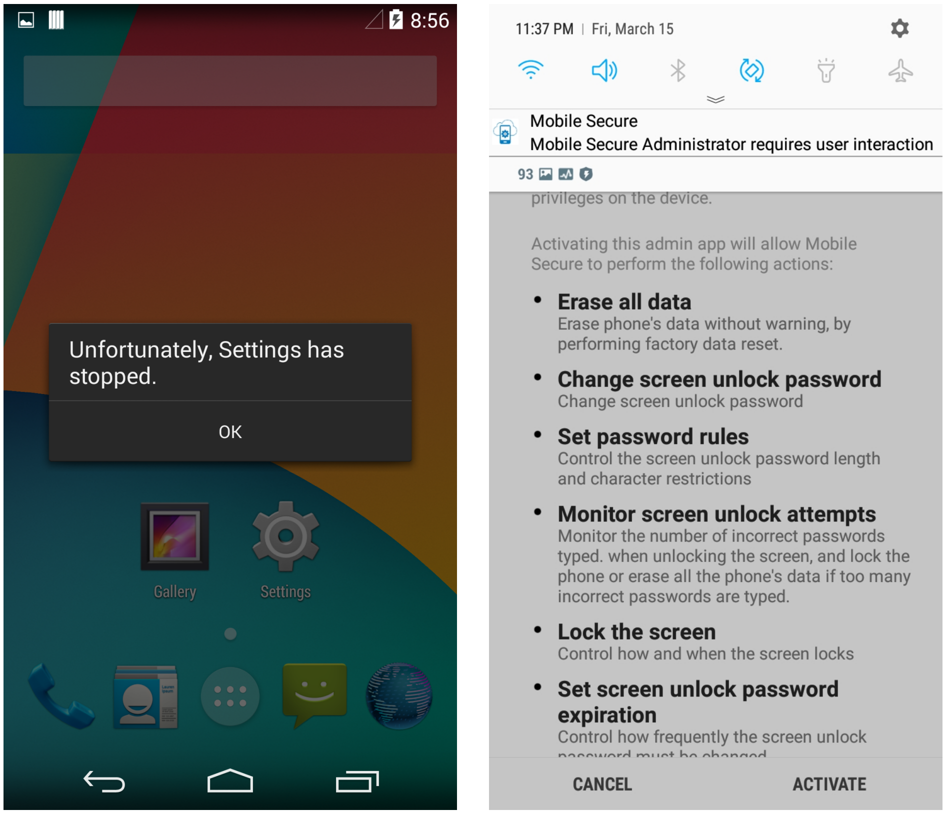
- Lock the screen
- Set lock-screen password expiration
- Set storage encryption

NO SIM CARD — EMERGENCY CALLS ONLY

Figure 5.2 *Capsule* Warning (left); locked screen (right).

Example 1. *Sberbank Online* disguises itself as an online banking app (for Russian bank Sberbank), to steal user credentials. The malware asks for administrator privileges upon installation, which, if permitted, can inflict serious harm to the victim's device. The app can also intercept SMS messages and incoming calls which could be a step to sidestep the bank's OTP (One Time Password) requirements. The app becomes "deathless" by preventing users from deactivating the app's DA privileges. As a result, the user *had no chance to see and heed the warning dialog and to permit DA deactivation.*

Example 2. *Check Point Capsule Connect*, a VPN app [53], was found as DDA Reset in December 2018, reported to Google in February 2019, and removed from Google Play around March 2019. As shown in Figure 5.2, when deactivating DA, the app pops a warning dialog and locks the phone with an unknown password.



Unfortunately, Settings has stopped.

OK

Mobile Secure
Mobile Secure Administrator requires user interaction

93

privileges on the device.

Activating this admin app will allow Mobile Secure to perform the following actions:

- **Erase all data**
  Erase phone's data without warning, by performing factory data reset.
- **Change screen unlock password**
  Change screen unlock password
- **Set password rules**
  Control the screen unlock password length and character restrictions
- **Monitor screen unlock attempts**
  Monitor the number of incorrect passwords typed. when unlocking the screen, and lock the phone or erase all the phone's data if too many incorrect passwords are typed.
- **Lock the screen**
  Control how and when the screen locks
- **Set screen unlock password expiration**
  Control how frequently the screen unlock password must be changed

Figure 5.3 *Mobile Tracker* crashes *Settings* app (left); *SAP Mobile Secure* pop-ups (right).

Unfortunately, to recover access to the device, the user must perform a factory reset.

Example 3. *Mobile Tracker* is a popular app (1,000,000+ installs) [106] that we detected as DDA Reset. We reported it to Google in February 2019; its DA capabilities have been removed while this paper was being prepared. The app can track device activity, delete files when the device is lost/stolen, etc. When disabling DA, clicking the deactivation button renders the Settings app unresponsive for extensive periods; eventually the app pops up a window finally asking the user to confirm the DA deactivation. Even if the user selects OK, the DA checkbox is still checked, and the Settings app crashes, which is shown in 5.3. After restarting the phone, *Mobile Tracker*'s DA checkbox remains checked. We found that the app keeps verifying the "checked" status of the DA checkbox.

DDA Hide. Apps in the DDA Hide category hide themselves from the DA list in the *Settings* app, i.e., the user cannot even see that the app is operating as a DA.

Example. *Bandwidth Meter* monitors network connections and displays Internet speed. When the user attempts to uninstall the app, Android shows a message that the app cannot be uninstalled, as the app is DA. However, the app does not appear in *Settings*' DA app list. The *Hidden Device Admin Scanner* app by Trend Micro failed to find this app. While Trend Micro's *Mobile Security & Antivirus* flags *Bandwidth Meter* as a potential unwanted app, it does not remove the app (it can only find it). This hiding behavior is caused by a security vulnerability in the *Settings* app, which omits to show a DA app in the list. Specifically, when updating the DA app list, the *Settings* app will first get the list of all activated DA apps and the list of all enabled DA. Only when an app is in *both the Activated and Enabled* lists, the *Settings* app will show it. However, some apps can be activated without being enabled, and use this artifact to hide from the *Settings*' DA app list.

DDA Expert. DDA Expert apps could in principle be deactivated but doing so is difficult for the average user. For example, malware Dowgin modifies the appearance of the check box in the DA list to disguise the fact that the app is still DA-active after deactivating the DA. The user can click the check box again to deactivate the DA. In actuality, this activates DA again. However, the user does not realize this trick and assumes that the DA cannot be deactivated.

Example 1. *MaaS360 Mobile Device Management* is a cloud-based mobile device management app (1,000,000+ installs) [100]; we found DDA Expert behavior which

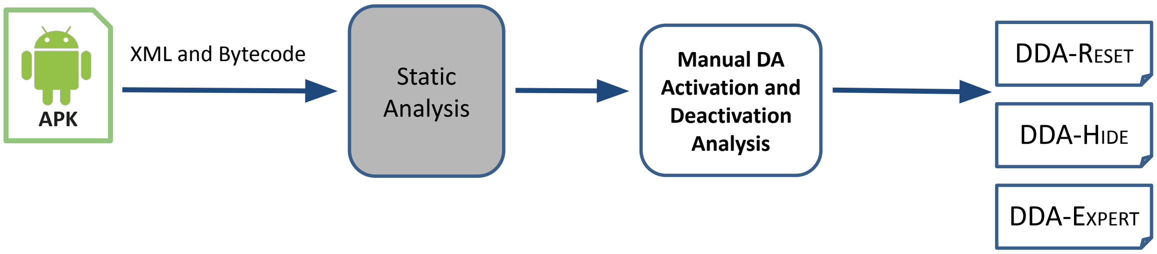


Figure 5.4 Overview of our workflow.

we reported to Google; the behavior has been corrected. Essentially when a user attempted to deactivate DA after multiple presses, *Settings* crashed, deactivating eventually. While it may not seem serious at a glance, many users have recently reported and vehemently complained of the inability to uninstall the app.[3] We found that *Settings*' crash is caused by *MaaS360* sending the intent DEVICE ADMIN DISABLED to *Settings* on a continuous basis.

Example 2. *SAP Mobile Secure for Android* is a device management client. As of March 17, 2019 the app is still available on Google Play [84], with DDA behavior still present. The app becomes "resistant" once the user attempts to deactivate DA. Specifically, as shown in 5.3 (right), upon deactivating DA, the app keeps popping up a notification with sound and shows the DA activation dialog, which forces the user to re-activate DA. Even after restarting the phone, the notification and sound resume. Only when DA is re-activated, the notification and sound would stop.

5.4 Methodology

We now describe our workflow, testbed, and Ground Truth procedure used for discovering and confirming DDA which we illustrate in Figure 5.4.

[3] Excerpts from two recent reviews on Google Play: (1) *"This freaking app locked my phone completely cant even use my own home screen or my other apps is in emergency mode wont let me do a thing how in the hell do I remove it???"* (2) *"not letting me uninstall a single thing in my phone. I cant even deactivate this app to uninstall it lmao. Its dictating what i can and cant have on my phone... dont download it"* [100].

Automatic Analysis. All 39,459 apps' manifests were checked for the BIND DEVICE ADMIN permission using two separate procedures: (1) using our static analysis tool [4] and (2) using `grep` on the Manifest extracted via Apkanalyzer.[5] We confirmed that we could check for the permission using at least one of these methods; this yielded 4,135 apps. We then used the static analysis tool to determine DA behavior.

Manual. We manually confirmed suspected DA behavior by:

- Attempting to deactivate DA, restart Settings and check whether DA is still deactivated, revealing DDA Reset and DDA Expert apps.
- Checking those cases where an app had the DA permission but was not appearing in the Settings' list, revealing DDA Hide apps.

This split the 4,135 apps into 3,350 that were not DDA and 785 that were actual observed DDA, or reported DDA by the tool.

Replication testbed. All behaviors, either revealed by the tool or manually (in the 785 apps), were verified on at least three phones from a five-phone pool: two Google Nexus 5s running Android 4.4.4 and 6.0.1, respectively; an LG G4 running Android 5.1; a Google Pixel 3 running Android 9; and a Galaxy J7 Crown running Android 8.

Ground Truth. Ground Truth is essential for finding False Positives or False Negatives, hence determining effectiveness. We determined Ground Truth via the automated and manual process described above (steps 2 and 3): we manually checked all 4,135 apps with DA permission, which was efficient due to batch processing and the fact that 3,350 apps were not DDA. The remaining 785 DDA or reported as DDA by our tool were subject to extensive analysis (step 3). The process yielded 578 true DDAs aka True Positives (with confirmed, replicated DDA behavior on multiple devices).

[4] The static analysis is not a contribution of this dissertation.

[5] `https://developer.android.com/studio/command-line/apkanalyzer`

5.5 Results

We performed a longitudinal study to measure the effectiveness of each DDA attack vector, and see how the Android version influences (permits or prohibits) DDA manifestation. While an app was deemed DDA if it could be confirmed on at least three devices/versions, here we only focus on apps that could be installed and run on *five* Android versions (5.1, 6.0.1, 7.1, 8.0, and 9.0), i.e., March 2015–August 2018, as this allows us to make more conclusive longitudinal observations. This stronger selection criterion reduced the number of apps to 301 malicious and 42 benign apps (compared to 468 malware and 110 benign apps for the three-version setup) for two main reasons. First, many apps were designed for late versions of Android hence failed to install on early versions; e.g., if the manifest specifies android:minSdkVersion=23, which corresponds to Android 6.0, the app will not install on Android 5.1. Second, old versions of apps that still run on Android 5.1 would immediately force an upgrade when started on 8.0 or 9.0; however, allowing the upgrade would violate our requirement to run the same APK on all five OS versions.

Figure 5.5 shows DDA prevalence: percentage of apps exhibiting that behavior in a particular Android version. We begin with several general observations. First, DDA Expert was the most prevalent behavior for both malicious and benign apps, respectively. Second, only three malicious apps employed DDA Hide; no benign apps employed it. Third, as DDA Reset apps cannot be uninstalled, these techniques are very rarely employed by benign apps (one app, *Check Point Capsule Connect*). We now make several longitudinal (evolution) observations.

Benign apps. Among benign apps with DDA Reset, the behavior is the same regardless of the Android version. However, DDA Expert apps have decreased with more recent Android versions. This is due to app behavior being ameliorated by the OS version, as explained shortly.

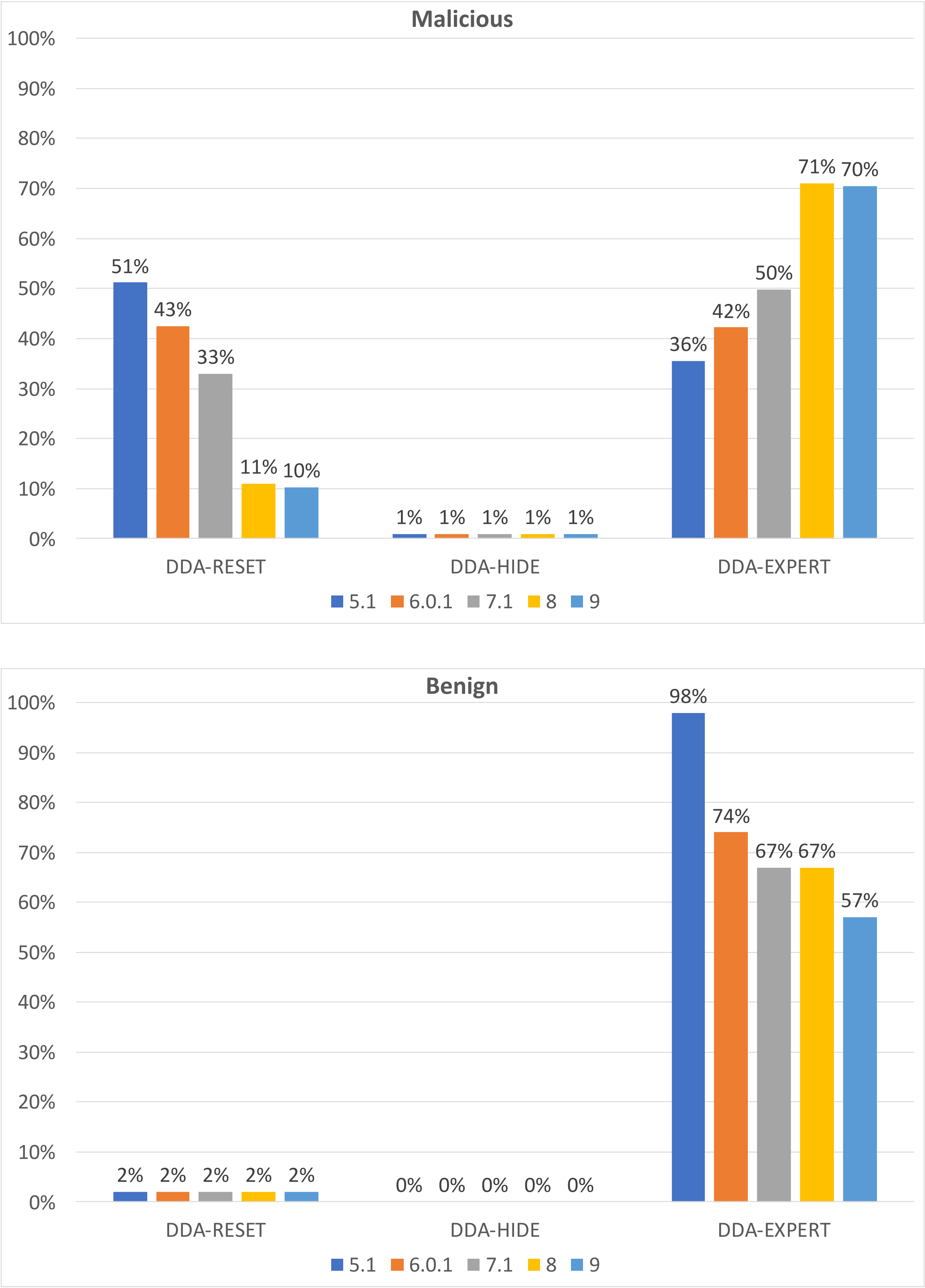


Figure 5.5 DDA prevalence across five Android versions in malicious apps (top) and benign apps (bottom).

Malicious apps. DDA Reset was prevalent in older versions of Android until Android 7.1 where the trend switched to DDA Expert. This is due to the following reasons that are purely OS oriented. Apps with DDA Reset in older versions often had an overlay that would prevent the user from, powering down the phone, accessing Settings, or the device itself, thus blocking the user from initiating any further activities and forcing the user to perform a factory reset. However, starting in Android 8.0, for some apps with an invasive overlay, a notification in the Notification Drawer [38] warns the user that the specified app has access to draw over other apps. When opening the notification, the system sends the user to the Settings option for the app hence allowing the user to disable the app's ability to draw over other apps and ultimately bypassing the invasive app overlay. Once that happens, the user is able to deactivate the DDA and uninstall the app. This is why in later versions of Android the DDA Reset behavior for some apps becomes DDA Expert.

In addition, starting in Android 7.1 the user has the ability to uninstall a DA app directly within Settings (not just disable DA). In older versions, some apps with DDA Reset would incessantly keep popping up the Settings option to activate DA once the user would deactivate the DA. With no other way to remove the app and stop the harassment, the user would end up having to factory-reset the phone. In Android 7.1 and later, however, once the behavior manifests, the user can uninstall the app immediately once the Settings option pops up. Despite improvements in the Android OS, DDA Reset behavior still exists, albeit in smaller numbers.

5.6 Conclusions

Overall, we characterized and quantified the legitimate, as well as the nefarious, use of DA capabilities in Android apps. Based on these observations we have constructed a static analyzer that exposes potential DDA behavior in a given Android app. We ran static and dynamic analyses on large corpora of benign apps and malicious apps.

We have revealed potential issues in more than 500 apps; the confirmed issues have been reported to Google's Android Security team. Our study and tool can improve Android security by helping end-users, developers, and app marketplaces analyze DA behavior. We have revealed examples of deceptive behavior among apps and clear lapses in mobile security which threaten users. We will now change focus, while keeping user interests at hand, and look further into the vast app landscape of Google Play, namely the Health & Fitness and Medical categories, and perform a comprehensive characterization of these apps revealing their functionality as well as potential risks and claims provided.

CHAPTER 6

HIGH LEVEL MEDICAL APP FRAMEWORK

There is a proliferation of medical mobile apps: Google Play alone has thousands of apps in the “Medical” category. Many such apps perform critical tasks (e.g., are used with a medical device or in lieu of a device); handle sensitive patient related information; perform diagnosis; or treat diseases. However, there are wide gaps between an app’s claims and users’ expectations as well as between app implementations and regulatory frameworks’ mandates. This chapter presents our analysis and characterization of 2,215 Android apps. We begin by introducing an automated classification scheme that integrates textual information extracted from multiple sources to establish the purpose and target audience for an app, based on fine-grained traits and high-level categories; we found that the most common functionalities involved connecting to medical devices (e.g., hearing aids, glucometers), offering tele-health services, or patient management. We then dive deeper into app nature and characterize it according to the function and domain of the app. We reveal actionable findings found in various facets of medical applications, regulatory frameworks and user privacy and safety.

6.1 Introduction

Over the past decade, digital/mobile health has been an area of mobile computing that has been developing as devices have become more advanced and more ubiquitous [51]. On the Android platform alone, this pervasiveness has led to tens of thousands of apps in the Health & Fitness and Medical categories. Furthermore, virtually all hospitals have enabled patients to access their health information via portal apps in both the outpatient and inpatient setting [85].

Many users of these medical apps are unfamiliar with the app landscape and unsuspectingly trust that the apps are safe. Users should not be expected to judge the legitimacy of a medical app or know what the app is doing with their personal data. Medical apps can be valuable tools but there is no universal standard for effective app checking, e.g., to not put personal data at risk. A 2015 study on apps that evaluate symptoms for self-diagnosis and triage reveal that many deficits exist in both aspects [125]. Such lapses in accuracy are potentially an issue of public health since these apps are often used to make healthcare decisions. Moreover, with the widening scope of medical apps, their capabilities and intended audience remain unclear. For example, the app *Instant Heart Rate: HR Monitor & Pulse Checker* has over 10,000,000 installs; the app's description states that it is the "most accurate" heart app and has been used in research. While the functionality is legitimate, the disclaimer states that "Instant Heart Rate should be used for entertainment purposes". In order to triage potential app abuses or misleading claims, a clear consistent classification scheme of apps and their functionalities is essential. In this joint work,[1] we present a characterization of medical apps. Using a dataset of 2,215 apps, we begin by categorizing apps using a multifaceted analysis employing three main sources from an app's metadata: app description, XML (extensible markup language) assets, and image assets (Section 6.2). By this process, we extract relevant medical keywords. The keywords are used to determine orthogonal fine-grained traits that describe app functionality (e.g., sending patient data, found in 775 apps, or handling insurance, found in 376 apps). Combining these traits leads to a higher-level categorization scheme that allows us to better understand the app's intended audience. We establish six categories: virtual visit, patient portal, medical device, professional, reference, and patient. Virtual visit apps allow users to interact with medical professionals

[1] Joint work appeared as: *Characterizing Medical Android Apps* by Raina Samuel, Iulian Neamtiu, Sydur Rahaman, and James Geller at the 16th Multi Conference on Computer Science and Information Systems (E-Health) 2022 in Lisbon, Portugal.

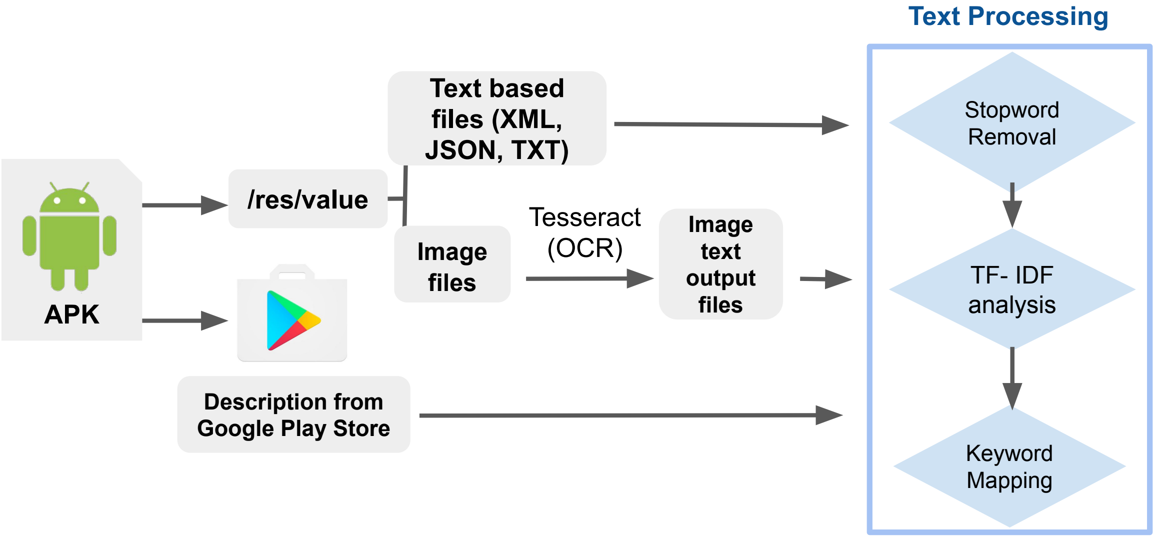


Figure 6.1 Trait extraction.

remotely. Patient portal apps allow users to access information regarding visits or book appointments. Medical device apps interface with an external device, such as a glucometer. Professional apps are intended for medical professional uses in office management or patient care. Reference apps provide study and reference material. Finally, patient apps are intended for general personal health reminders. We find that the most popular category is patient, with 1,993 apps, followed by reference with 1,590 apps. Our classification scheme shows that the most common functionalities of medical apps involve connecting to medical devices, tele-health, and medical calculators. In Section 6.3 we discuss actionable findings from our research. We investigate possible lapses found in the way regulatory agencies approve and determine medical apps and their functionalities. We discuss privacy implications of handling user data and how developers and marketplaces should be more transparent in how sensitive data is handled.

6.2 Characterization

App characterization – understanding the nature, purpose, and target audience of an app – is challenging, as detailed next. To address these challenges, we use multi-source information along with a multi-rater human approach. First, we use information retrieval to extract terms of interest. Then we define first-order low-level *traits*. Building upon traits, we establish high-level *categories*.

Challenges. Characterization is a major challenge for several reasons. First, apps may serve more than one purpose, e.g., an app may manage a patient's prescription, help locate the nearest emergency room, and support video chats with a provider. Second, such features are hard to automate and detect automatically (e.g., video chat can be home-made as opposed to using a video chat library). Similarly, location/mapping services can serve multiple purposes so the presence of such a library simply indicates that the app provides location-relevant services. Third, the app description on Google Play is at the developers' latitude and can be incomplete, inaccurate, or misleading (e.g., we found "medical prank" apps that are trying to pass as legitimate apps). Fourth, actual app functionality can only be reconstituted from heterogeneous sources via a multi-faceted analysis of app description, embedded images, app bytecode, etc. We started by retrieving all apps from Google Play's Medical category along with their descriptions. We only retained those apps which had English descriptions and at least 1,000 installs, for a total of 2,215 apps. The design of our approach for extracting relevant text is shown in Figure 6.1. Next we describe the sources, methodology, and characterization results.

6.2.1 Sources

An APK is essentially an archive of directories and files that include bytecode, resource files, assets, and libraries. Among all these files, we find that text relevant to app functionality and user interactions with the app appears in three main locations: XML assets, images, and app descriptions, as shown on the left of Figure 6.1. We describe each of these and provide evidence why all three sources are needed.

XML Assets. Layout XML files, stored in the app's `res` directory, define the user interface by storing all text views, buttons, and other UI elements. We are interested in these features as we can discover what information the app is requesting from the user and what kind of information the user provides to the app by interacting with it. String XML files store strings accessed by the application, which constitute another key location of medical terminology that can be extracted.

Image Assets. Image assets, also stored in the app's `res` directory, are relevant as well. For example some apps may opt to use an image as a button rather than using it in an XML asset. In other cases, images may have text included making it important to be analyzed as well. We use the Tesseract OCR (optical character recognition) package [136] on image files to extract English text present in images.

Descriptions. App descriptions are found on Google Play and not within the app itself. As a result, the description of an app allows a user to understand what an app does prior to installation. As the description is the first impression a user has of the app, its functionalities should be clearly defined in a way that help users establish an app's purpose confidently and securely. However, as the description is usually written by the app developers themselves, the app is often portrayed in a flattering and overly positive way to attract users. Therefore, app descriptions can be inaccurate, misleading, or incomplete, which we found to be another essential aspect of an app that should be considered in our analysis.

Why all three sources are necessary. Table 6.1 illustrates why using only one of the three aforementioned sources is insufficient: the table shows, for three apps, where the relevant keywords are located. For app *SimplePharmacology*, 69% of the relevant keywords are in the image assets while the description contains no relevant keywords whatsoever. In contrast, for app *MyNM*, all the keywords are found in the

XML assets; images and the description contain no relevant terms. Finally, for app *AnthroCalc*, all keywords are in the app description. Therefore we need to analyze and integrate information from all three sources.

Table 6.1 Location and Frequency of Relevant Keywords

App	**Frequency (%)**		
	XML Assets	Image Assets	Description
SimplePharmacology	31	69	-
MyNM by Northwestern Medicine	100	-	-
AnthroCalc	-	-	100

6.2.2 Methodology

In order to create a clear classification scheme based on app functionality, we referred to ICD-11 (International Classification of Diseases) [10] and PHI (Protected Health Information)[78] terms. ICD codes provide a reliable established standard of diseases and health conditions. PHI terms allow us to obtain a broad idea of what data is required from users in certain apps to determine their functionalities. We began our text processing with extracting the descriptions. First, the descriptions had stop words removed to focus on conceptual information in the text, followed by a TF-IDF (term frequency – inverse document frequency) analysis [17] based on ICD keywords and PHI terms. As a result, we could provide a preliminary classification of each application based on keyword matches and frequencies. The process was repeated for XML and image assets. With the resulting keywords extracted, we observed which resources provided the most relevant results. More keywords were found in the XML files of apps as opposed to image files (via Tesseract), or app descriptions. This evidences that descriptions do not paint a complete picture.

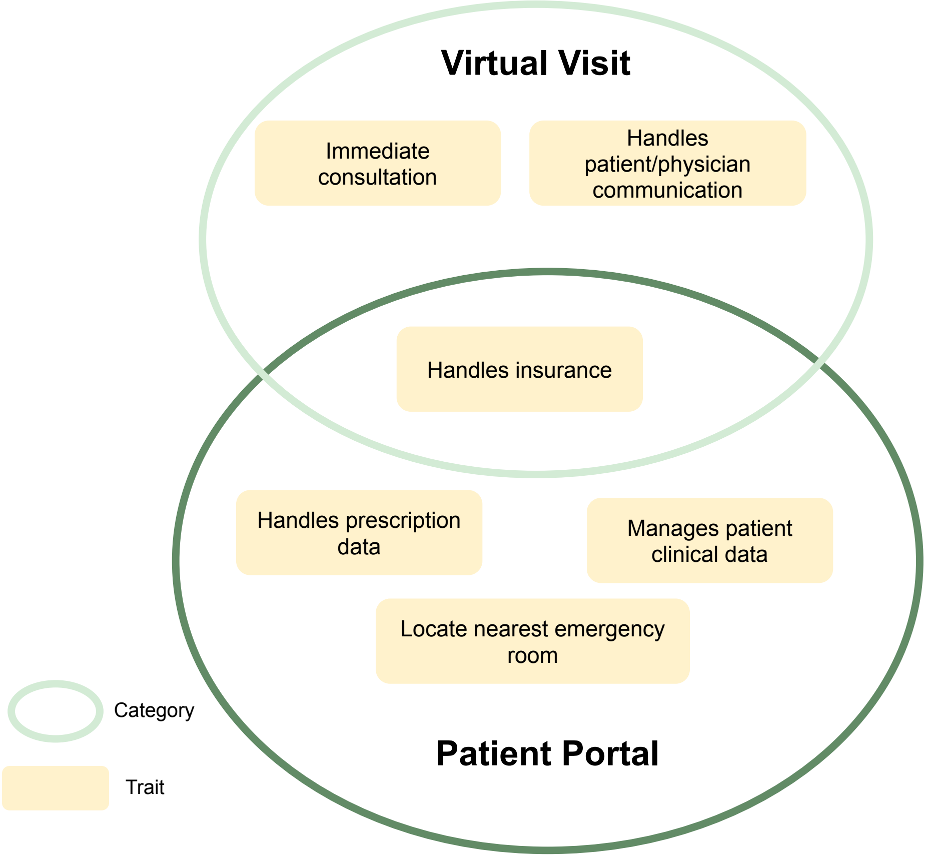

Virtual Visit
Immediate consultation
Handles patient/physician communication
Handles insurance
Handles prescription data
Manages patient clinical data
Locate nearest emergency room
Patient Portal
Category
Trait

Figure 6.2 Category determination based on traits.

Defining Traits and Categories. We employed a multiple-raters approach [71] to determine traits and categories: three human raters had to come to 100% agreement on what constituted and differentiated the various traits. Raters had to agree first on what should be considered a unique trait of a medical app and which keywords should be used in determining that trait (traits essentially define low-level orthogonal functionality "facets" for apps). As a result, 19 traits were determined. Subsequently, raters would then agree on what combination of traits would dictate the category of an app. Once the baseline was set, the app was classified using traits into categories, as illustrated in Figure 6.2. In this way, some apps may have multiple traits and various categories. In this way, some apps may have multiple traits and belong to various categories. However, such categorization provides a more nuanced view on the general functionality of certain medical apps. We now discuss our findings.

Table 6.2 Most Common Traits Found in Apps

Trait	**Description**	**%Apps**
Anatomy	Anatomy reference material	61%
Well-being reminder	Keeps track of patient habits (e.g., sleeping, water intake, exercise)	51%
Medical student study aids	Exam and practice questions for medical students	38%
Medical calculator	Calculates readings without saving patient information	37%
Sends acquired patient data	Sends inputted patient data such as blood sugar etc. to a provider	35%
Handles prescription data	Patient information regarding prescriptions and treatments	28%
Manages patient clinical data	Stores the medical history of a patient	27%
Visual guide	Visual reference material used by students and medical professionals	26%
Medical procedures	Procedural reference material intended for medical professionals	22%
Patient journal/diary	Patient behavior or progress combating disease or nutrition	20%
Disease name	Disease reference material used by medical professionals or students	20%
Handles patient/physician comm.	Stores and transmits patient information between patient and provide	20%
Patient symptom tracker	Keeps track of various symptoms, may lead to diagnosis	19%
Drug name	Drug name and pharmaceutical reference material	18%
Handles insurance	Stores patient medical history related to insurance policy	17%
Immediate consultation	Virtual consultations with a provider	16%
Dose calculator	Calculations for patients and providers to administer medication	16%
Locate nearest emergency room	Using current location to find an ER	14%
Device measuring patient data	Using an external device to collect readings such as blood pressure etc.	14%

6.2.3 Traits

Traits are defined as single aspects of app functionality, orthogonal to other aspects. Apps can exhibit multiple traits, as apps can provide several functionalities. We determined traits by finding common ICD terms and medical keywords. As a result, we defined 19 unique traits; their definitions and frequencies are shown in Table 6.2. Many of the traits are self-explanatory and commonly used, such as ‘Anatomy’ and ‘Locate nearest emergency room’. There are certain traits that needed further refinement, specifically those dealing with patient data management. While creating and characterizing these traits, we also evaluated the potential risk that apps with certain traits have versus apps which do not. Table 6.2 reveals that many common traits involve reference material for medical professionals. For instance, Medical Student Study Aids are found in the top five traits in medical apps, as well as medical calculators, which are often used by professionals.

Table 6.3 Category Determination From Traits

Category Trait	Reference	Patient Portal	Professional	Patient	Virtual Visit	Medical Device
Sends acquired patient data			●			●
Handles prescription data		●		●		
Handles patient/phys. comm.			●	●	●	
Handles insurance		●		●	●	
Manages patient clinical data		●	●			
Device measuring patient data						●
Patient symptom tracker				●		
Well-being reminder				●		
Dose calculator			●			●
Patient journal/diary				●		
Immediate consultation			●		●	
Anatomy	●		●			
Medical student study aids	●		●			
Medical calculator			●	●		
Disease name	●					
Medical procedures	●					
Visual guide	●					
Drug name	●		●			
Locate nearest ER		●	●	●		
Total number of apps	1,590	327	509	1,993	1,269	609
Total percentage of apps	72%	15%	23%	89%	57%	27%

6.2.4 Categories

The various combinations of certain traits allow us to determine specific categories of medical apps as is evident in Table 6.3. We established six unique categories that apps may fall into.

Reference. These apps serve either as general references regarding medical terms or first-aid procedures. Some apps are study aids or provide quizzes for medical professionals in training.

Patient Portal. Users can schedule and make appointments with their medical providers and view their lab results or test results and data from their visits. In addition, users can search for nearby providers.

Professional. These apps are directed towards medical professionals ranging from medical staff to office assistants. Many apps help medical clinics with scheduling and handling patient data in a professional setting.

Patient. Apps in this category are aimed at patients to help them log their daily progress or daily habits such as sleeping or pill reminders.

Virtual Visit. These apps provide for virtual visits, e.g., via a video call with a medical professional. In doing so, users often provide personal information and discuss their symptoms.

Medical Device. These apps are considered as medical devices or work in tandem with devices, such as hearing aids, glucometers, or sphygmomanometers for hypertension. Apps in this category can be used to store device readings and be maintained as a log or can be used as a remote control for the device.

6.3 Actionable Findings

Our categorization has revealed that medical apps serve a broad audience and variety of purposes. However, because many such purposes are sensitive or even critical, and not intended for a general audience, there should be barriers for app access control. Theoretically, as these apps are free, and found on a public app distribution platform, anyone can download and use them, even though the apps are meant exclusively for professionals. Generally, apps that are meant for professionals in a hospital or clinical setting usually require credentials to access such systems. However, there are professional apps which can potentially result in a diagnosis or interface with a medical device for a procedure; if such apps are available for general use, it can lead to possible user harm. Hence there is a need for strong regulatory frameworks protecting end-users. We now describe actionable findings covering various aspects of medical apps. We review current regulations and definitions regarding mobile health and medical apps established by various legal entities throughout the world, while also finding certain lapses and difficulties in implementing these guidelines. From these definitions, we discuss potential privacy implications and user safety concerns.

6.3.1 Regulatory Framework Enforcement

Actionable finding: Regulatory frameworks should be clearer defined and more accessible for developers when creating medical apps managing user data. Medical apps can perform critical functions that involve patient data or other sensitive information. Overall, app users generally assume that apps are "certified" and trustworthy when making medical decisions. The question that arises is whether these apps are indeed approved by regulators and safe for use. For example, in the United States, the FTC (Federal Trade Commission) provides definitions and guidelines for app developers. The guidelines indicate whether the app is a medical device, or a medical app; as well as whether the FTC will apply any regulatory oversight [69]. Additionally, the FDA (United States Food and Drug Administration) regulates functions of mobile devices that use device sensors (camera, light, vibrations) to perform medical device functions (e.g., measuring blood pressure), connecting a mobile device to a medical device and being able to manipulate it from the mobile device (e.g., alter settings of an implant), or active patient monitoring (e.g., acquiring signals from a cardiac monitor) [64]. We further discuss US regulations and their scope in Chapter 9. EU regulation of mobile medical apps focuses on potential privacy concerns that may arise. Mobile health apps must comply with data protection laws (Data Protection Directive) that were enacted, as well as ensuring that apps provide "clear and unambiguous information about processing to end users before app installation" [60]. Some Asian countries, such as China and Japan, regulate standalone medical software as medical devices, though depending on the overall software class, whose definition is based on functionality [72]. Overall, regulatory bodies have general guidelines on medical app behavior and functionality. However, there is no clear standard for app developers to easily refer to when developing a medical app. Having an accessible flowchart or a streamlined explanation of definitions would aid developers as well as app markets (Google Play, Apple's App Store) in managing the apps, especially apps handling users' medical data.

6.3.2 User Security and Safety

Actionable finding: Medical apps should be more transparent regarding user data management prior to installation. App functionality plays a large role in determining whether the app falls under a regulatory framework. Medical apps often manage identifiable and private health information, that is, demographic information related to a user's health or condition that can be used to identify the user. For instance, in the US, if such apps work with health care providers or HIPAA entities, they are subject to HIPAA rules regarding security [78] and privacy [79] and what must be done when a breach has occurred [77]. However, not all data acquired by an app is considered identifiable health information. For example, an app measuring a user's weight and blood pressure is not considered a big security risk, compared to an app that tracks patient activity and prescriptions. Thus, certain apps pose lower risks to user privacy and would not need to be under scrutiny from regulatory bodies. An example would be apps that are general aids or of general purpose (e.g., magnifying glass); automate general office functions in healthcare and are not used for diagnosis; and educational apps (e.g., flashcards, encyclopedias, textbooks). These apps are neither regulated nor will have any discretionary enforcement exercised on them. However, as discussed previously, many medical apps handle patient data, and despite regulations and guidelines, users do not know how securely their data is managed or transmitted. App developers and markets must be more forthcoming and transparent about patient data management, by concisely explaining to users prior to installation what happens to their private health information. Potentially, these entities should be held accountable, should any leaks occur.

6.4 Conclusions

Medical apps across many categories have been implemented and publicized; these apps serve millions of users and provide a multitude of functions. To better understand the app landscape, our study categorizes medical apps based on stated and observed functionality. Overall, our research makes several contributions. First, we provide an automated approach and study that characterize medical apps into sub-categories to better understand their purposes and functionalities. Second, we observe the most common functionalities of medical apps. Third, we discuss regulatory frameworks and user privacy practices. By doing so, we are better equipped to undertake further studies into app behavior, app security, app claims, etc.; and ultimately improve the health and well-being of app users.

CHAPTER 7

CHARACTERIZING MEDICAL APPS

Now we will discuss how medical apps may over-claim and overstate their intended purposes. This chapter presents the classification of claimed app behavior based on ICD conditions and provides a categorization of misleading claims. To determine potential gaps between app claims and app behavior, as well as between app claims and user expectations, we conducted a study on over 2,000 Android apps. We first developed an information retrieval approach that maps an app's description to medical (ICD) terms, hence delineating the app's medical scope and stated goals. Next, based on app functionality, we categorize apps into (a) apps that measure or manage a physiological parameter, (b) apps that claim to treat conditions, and (c) apps for self-assessment.

7.1 Introduction

Due to the convenience and ubiquity of medical apps, users trust medical apps and generally assume that apps are validated and accurate. However, there is no direct evidence on whether a medical app is performing its claimed functions. For instance, in a study regarding blood pressure monitoring apps, users liked the perceived accuracy; however, the app under-reported users' actual systolic pressure and provided inaccurate results which gave users a false sense of security [113]. This joint work [1] involved us conducting a study on more than 2,000 Android apps collected from Google Play to understand (1) the medical conditions targeted by medical apps, and (2) the claims apps make, e.g., regarding diagnosis or cures; revealing and categorizing lapses between app claims and actual functionality. To define app

[1] Joint work appeared as: *Could Medical Apps Keep Their Promises* by Raina Samuel, Iulian Neamtiu, and Sydur Rahaman at the 16th Multi Conference on Computer Science and Information Systems (E-Health) 2022 in Lisbon, Portugal.

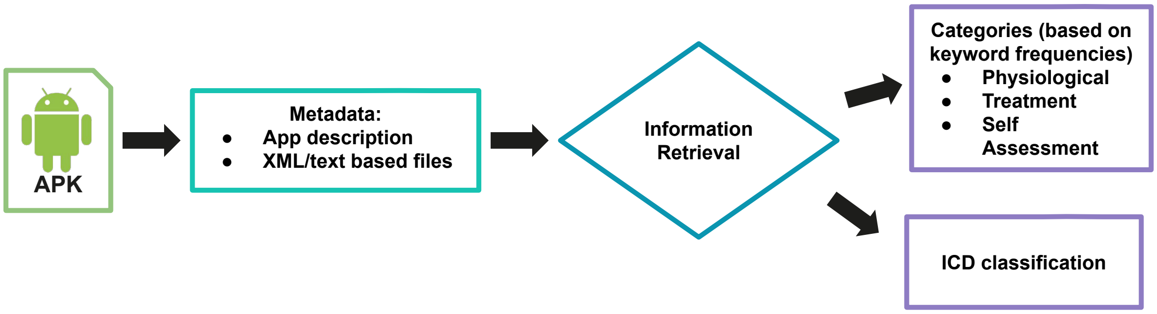


Figure 7.1 Overview of methodology.

behavior and nature, we mapped app metadata terms onto ICD-11 (International Classification of Diseases) codes. Using ranked retrieval text analysis, we were able to accurately shed light on common conditions apps may claim to treat or manage (Section 7.3). For apps that perform measurement and tracking, we found that most ICD codes were related to physiological management, such as weight loss or heart rate measurement. For apps that address conditions, we found that the most common conditions include **Elevated blood glucose level (MA18.0)** and **Speech therapy (QB95.5)**.

Next, we focus on exposing questionable claims found in app descriptions. We classified apps into three main categories of claimed behavior: physiological (Section 7.5), treatment (Section 7.6), and self-assessment (Section 7.7). Focusing on app descriptions allows us to better observe what may possibly convince users into installing certain apps. We established keywords and frequencies to categorize suspicious behaviors accordingly. Within each category, we investigated app claims and compared these claims with what an app can actually accomplish on a mobile device; we found a wide gap between claims and attainable functionality. Overall, we make the following contributions:

1. A classification of app behavior based on medical conditions established by international standards (ICD-11).
2. A classification of possible misleading claims found in Medical apps.
3. A discussion of app disclaimers and misleading description terms.

7.2 Methodology

We begin by describing our overall methodology illustrated in Figure 7.1. We acquired app descriptions and apps (APK files) from Google Play's Medical, as well as Health & Fitness, categories. This resulted in a total of 2,339 English language apps that had 1000+ installs. From these apps, we used their descriptions and relevant text-based app metadata to determine ICD codes and observe claimed app functionality and misleading claims. We map the medical conditions that apps may claim to treat (or monitor) onto an established ontology, ICD-11 codes. ICD – the classification of diseases used by the World Health Organization – provides an international standard for uniform naming of diseases and health conditions. Extracting ICD terms from app metadata not only enables us to identify possible conditions apps may claim to treat or monitor, but also (1) can reveal lapses in app descriptions regarding functionality and (2) help us understand further the general medical app landscape. App descriptions and text metadata were processed, removing stop-words and irrelevant terms. We will next discuss how we managed to extract ICD codes and categorize app functionalities via information retrieval. We categorize these apps based on claimed behavior mentioned in app descriptions and result in four main categories shown in Table 7.1.

Table 7.1 Categories of Problematic Descriptions

Category	**Number of Apps**
Physiological	430
Heart Rate Measurement	115
Optometry	98
Blood Sugar Measurement	87
Hearing Test	42
Skin Cancer	41
Body Temperature Measurement	31
Weight Loss	16
Treatment	320
Natural Home Remedy	200
Hypnotherapy/Brain Wave Therapy	71
Pain Relief	49
Self Assessment	500
Mental Health	309
Symptom Tracking	106
Pregnancy Quizzes	85

7.2.1 ICD code mapping challenges

We begin by describing the process and challenges faced when extracting ICD codes from apps. We used 106 ground truth apps as a basis to determine the score threshold for matched terms. The first challenge was determining relevant app metadata. Using irrelevant terms and certain stop words can result in inaccurate ICD mappings or even no matches. We compared extracted keywords between the app description and XML files in order to understand common medical app functionalities and which source would be the most effective in mapping with ICD codes. We found that XML files provided less relevant results despite having more medically related keywords, especially for apps used as reference material, intended for patients, or to interact with patient portals. This is because XML files contain more fragmentation and individual words rather than cohesive sentences to provide any meaningful input. The second challenge involved finding the best method to map and extract terms from app descriptions. We began with an initial mapping with a TF-IDF (term frequency – inverse document frequency) analysis, which showed that most apps correlated to the ICD code **MA13.1 (Finding of alcohol in blood)**. However, when we attempted a ranked retrieval text analysis, we found much more accurate ICD terms mapped to keywords.

Using a naive approach leads to inaccuracy in text extraction, and we show such discrepancies in Table 7.2. Here we compare the keywords extracted from TF-IDF analysis versus those from ranked retrieval; the text in red indicates inaccurate or irrelevant keywords that do not map to the accurate ICD code displayed. We see that ranked retrieval provided the most accurate results. Thus, we used ranked retrieval to obtain each app's ICD code.

Table 7.2 Keyword Discrepancies in ICD Conditions

App Name	TF-IDF	Ranked Retrieval	Category	ICD term
com.ebsco.dha	‘management’, ‘difficulty’, ‘disorder’, ‘condition’	‘health’,‘refer’;clinic’;care’	Patient Portal	QB10 (Medical services not available in home)
com.sonova.easyline.rcapp	‘fitting’,‘extent’, ‘period’, ‘carried’	‘control’,‘connect’, ‘hearing’, ‘remote’	Medical Device	QB31.4 (Fitting or adjustment of hearing aid)
com.pocketprep.nptepta	‘disease’, ‘specified’, ‘defect’, ‘vertical’	‘brain’,‘test’,‘therapy’, ‘nervous system’	Reference	MB72 (Results of function studies of the nervous system)
com.ninezest.stroke	’therapy’, ’devices’, ’malignant’, ’miscellaneous’	’stroke’,’therapy’, ’speech’, ’enhance’	Patient	QB95.5(Speech therapy)
com.srems.protocol	‘malignant’, ‘miscellaneous’, ‘classified’,‘harm’	‘region’,‘clinic’, ‘treatment’, ‘cardiac arrest’	Reference	MC82.1 (Bradycardic cardiac arrest)
com.easymobs.pregnancy	‘milestone’, ‘pelvic’, ‘muscle’, ‘certain’	‘help’,‘week’,‘care’, ‘pregnant’	Patient	MF34(Pregnancy symptom or complaint)
com.beltone.hearplusapp	’devices’,’general’, ’specified’, ’miscellaneous’	’manage’,’sound’, ’aid’,’tinnitus’	Medical Device	QB31.4 (Fitting or adjustment of hearing aid)
de.qurasoft.amsspiroapp	‘lung’, ‘pulmonary’, ‘eye’, ‘communicating’	‘asthma’,‘device’, ‘measure’, ‘symptom’	Medical Device	J45.8 (Asthma)
spm.nashres	‘cirrhosis’;score’, ‘voice’, ‘progress’	‘liver’,‘disease’, ‘histology’, ‘fibrosis’	Reference	DB92.0 (Non-alcoholic fatty liver disease)
com.usatineMediaLLC. dermoscopyTwoStep	‘need’,‘full’, ‘contain’, ‘pattern’	‘melanoma’, ‘carcinoma’;diagnosis’, ‘treatment’	Professional	2C31.Z (Cutaneous squamous cell carcinoma)

7.2.2 Categorizing claimed app functionalities

Next, we will discuss how we categorized app behavior. Here we focused solely on app descriptions, as they are the initial reasons why users download apps. We used 33 apps as ground truth, which we had manually determined as potentially misleading due to specific terms in their descriptions. We focused on generic terms such as “diagnosis,” “entertainment purposes,” “instant,” and “camera” and applied a TF-IDF analysis on the full dataset, resulting in a subset of 1,250 apps matching these criteria. Once the subset was established, we then manually reviewed common patterns based on general functionality to create a categorization. We developed another set of keywords to categorize the 1,250 apps into three categories using TF-IDF analysis. Finally, to better refine our categories and the broad functions we found, we further characterize

Table 7.3 Top 10 ICD Codes Based on Matched Medical Terms in App Descriptions

ICD Code	ICD Title	#Apps	Use
MG43.5	Excessive weight loss	511	Weight Control
MC82.1	Bradycardic cardiac arrest	255	Heart Rate Measurement
QB30.3	Adjustment or management of vascular access device	242	Pacemaker Management
QF21	Fitting or adjustment of hearing aid	232	Hearing Aid
MA18.0	Elevated blood glucose level	135	Diabetes Management
M54.5	Low back pain, unspecified	120	Pain Management
QA41	Pregnant State	104	Pregnancy Tracking
CA23	Asthma	84	Asthma Management
QB95.5	Speech Therapy	75	Speech Aphasia Treatment
H93.1	Tinnitus	70	Hearing Aid

them into more specific subcategories. In doing so, we reveal possible lapses in claimed behavior and their legitimacy, especially in popular apps.

7.3 Medical Conditions

We will now present our findings. First, we will discuss the results of the ICD code analysis and the top codes found. Then we will describe the claimed app behaviors found in app descriptions.

7.3.1 Top ICD Conditions

ICD codes extracted from app descriptions help us ascertain whether descriptions accurately describe/explain app functionality. Table 7.3 displays the top ICD codes along with an explanation of how it is used and its categorization. We found that most of the ICD codes relate to weight loss apps due to the frequency of the term: **MG43.5 (Excessive weight loss)**. We also see many ICD codes related to apps which connect to external medical devices, especially with pacemakers (**QBB30.3: (Adjustment or management of vascular access device)** and hearing aids (**QB31.4: Fitting or adjustment of hearing aids**). Among the top ICD codes, we found very few apps for professional use relate to any, if at all. This is because many app descriptions related to professionals or clinicians are either very vague or too complex to map correctly to a single specific ICD code. Nevertheless, we were able to accurately map ICD codes to medical conditions in apps that claimed to treat said diseases.

7.4 Claimed App Behavior

We characterized app functionalities into three main categories: Physiological, Treatment, and Self-Assessment. Apps in these categories are examples of behavior that may potentially require regulations or further scrutiny and exemplify the need to categorize claimed app functionality. Apps should be clearer about their true functionalities in their descriptions while being explicit in their disclaimers; many times, disclaimers are hidden in the text or towards the end of the Google Play description; when the description is lengthy, users may end up ignoring or missing the caveat completely. We will now describe each category and subcategory found in Table 7.1 starting with Physiological, Treatment, and Self-Assessment.

7.5 Physiological

Apps in this category claim to be able to measure certain physiological parameters such as heart rate or blood pressure, using the camera and other smartphone sensors. Concerningly, these apps claim to provide some form of diagnosis based on the measurement; furthermore, the apps claim that their measurements are accurate. We have found 430 such apps, categorized as follows.

Heart Rate. Heart rate-measuring apps use the smartphone camera's flash feature to measure a person's pulse. Measuring heart rates via a smartphone camera is not inherently inaccurate or deceptive, though a study has found differences between results obtained with apps versus results gather via clinical monitoring [58]. However, users should not solely rely on such apps for diagnosis or treatment. For example app, *Cardiac diagnosis (arrhythmia)*, with over 1,000,000 installations, states no disclaimers or recommendations to seek a medical professional or use an actual heart monitor along with the app. The accuracy is generally unknown, especially how the app manages to detect such conditions. Unless these apps work in conjunction with an external medical device, such as a blood pressure meter or heart monitor, the accuracy of such apps should not be relied on for diagnosis. Moreover, we believe that (1) such apps should include a disclaimer or recommendation to consult a medical professional, and (2) the term 'diagnosis' should be removed from apps' titles.

Optometry. Optometry apps claim to measure vision acuity by providing eye exams testing for astigmatism, near and far-sightedness or color blindness. While these apps may provide a basic benchmark for vision, without a medical professional's diagnosis, the apps should not be used as a sole medical opinion. As a result, all apps with this functionality must include a recommendation to report their results to qualified ophthalmologists or optometrists before taking any sort of action.

Blood Sugar. Blood sugar apps claim to measure or track blood sugar. While many of these apps do have this behavior, as they work with a glucometer, many do not – the apps simply serve as a journal. Apps claiming to measure or track blood sugar without connecting to a glucometer or any sort of device can be misleading. Additionally, some apps whose name contains "Blood Sugar Test" have disclaimers stating the app cannot measure blood sugar but provides information on how to manage diabetes. Thus, these apps should modify their titles to better reflect app functionality, e.g., "Blood Sugar Tracking" or "Blood Sugar Log".

Hearing Test. Hearing test apps are different from hearing aid apps, which tend to connect to an external hearing aid device, serving as a remote control. These apps claim to provide (1) tests regarding tinnitus and (2) therapies for hearing issues;

nevertheless, users need to see an ENT or audiologist for a reliable and accurate diagnosis.

Skin Cancer. Skin cancer apps use the device's camera to take pictures of skin and then use an AI algorithm to provide a preliminary diagnosis regarding skin cancer. Apps claiming to detect skin cancer solely through a device's camera and without a blood test are deceptive and misleading. An example would be the app *Medgic* which uses AI to check for dermatological conditions or diseases by using the device's camera. While AI algorithms have been able to detect conditions before, prognoses cannot be solely confirmed by a simple photo of one's skin – other tests must be administered in order to make a conclusion. The app's description contains a disclaimer, albeit at the end, stating how the app is not a replacement for medical advice and that not all results are 100% guaranteed. Another app, *Visus*, states that it is an experimental app that is publicly deployed and that its algorithm is "30% more sensitive and precise than a conventional board-certified radiologist".

Body Temperature. Body temperature apps claim to measure users' temperature, e.g., to detect a fever. However, this is ultimately misleading, as mobile devices do not have any means to measure temperature in their sensors. Instead, these apps serve as a mere journal to track user-inputted values for body temperature.

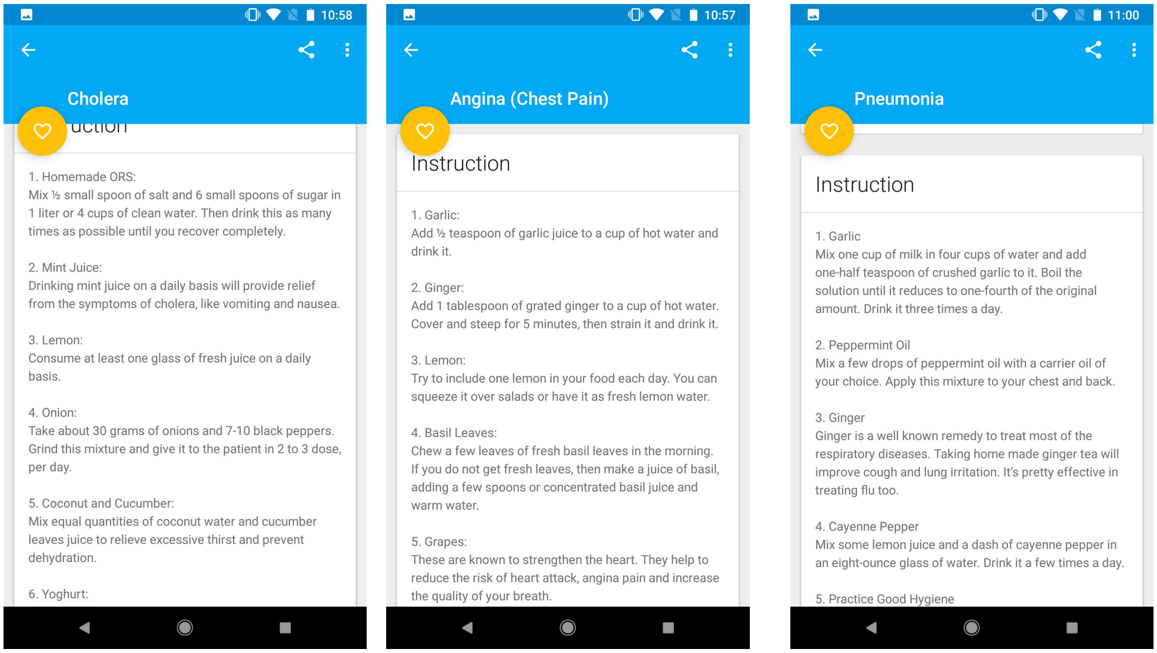
Weight Loss. Weight loss apps are numerous by nature, as seen in our ICD mapping. However, in this specific categorization we focused on apps that over-promise results within an arbitrary or unrealistic time frame or even "instant" results. We found that many apps do not urge the users to seek medical opinions prior to attempting weight loss. For example, the app. *Lose Weight Fast at Home - Workouts for Women*, with over 1,000,000 installs, claims that users following the app's regimen will lose weight in 30 days. However, there are no mentions in the app description of the influence other crucial factors such as diet, water intake, or genetic factors, have in weight loss.

7.6 Treatment

These apps claim to be able to cure diseases. We have found a total of 320 apps, falling into several subcategories.

Hypnotherapy/Brain Wave Therapy. These therapies are complementary forms of medicine (i.e., used to supplement traditional treatment methods). Apps in this category tend to not mention the importance of standard or clinically proven medical treatments to be used in conjunction with their suggested therapies. Hypnotherapy results are generally not clinically proven and may have adverse effects on users who are prone to epilepsy or other neurological conditions [73]. For example, the app *Atmosphere: Binaural Therapy Meditation* which has over 500,000 installs, states that it is able to "heal your DNA" with its guided breathing and meditation; nevertheless, the app description contains a disclaimer that the app is only for "entertainment purposes" and should not be a substitute for medical treatment.

Natural Remedy. These apps provide references to natural remedies, e.g., certain herbs or foods, to manage and treat specific diseases, such as skin diseases or even cancer. They also claim to help users "self-cure" certain conditions. Although apps that provide home remedies are not malicious or intentionally misleading, they should never be a replacement for actual treatment prescribed by a medical professional. While there are natural remedies to basic non-life-threatening illnesses or wounds, an app is not an alternative to prescribed treatment from a medical provider. For example, the app *Doctor at Home*, which has over 100,000 installs, claims it can provide treatment for "110 diseases" and "cure diseases at home". Examples of three conditions – cholera, angina, pneumonia – and app-prescribed "cures" are shown in



Cholera

uction

1. Homemade ORS:
Mix ½ small spoon of salt and 6 small spoons of sugar in 1 liter or 4 cups of clean water. Then drink this as many times as possible until you recover completely.

2. Mint Juice:
Drinking mint juice on a daily basis will provide relief from the symptoms of cholera, like vomiting and nausea.

3. Lemon:
Consume at least one glass of fresh juice on a daily basis.

4. Onion:
Take about 30 grams of onions and 7-10 black peppers. Grind this mixture and give it to the patient in 2 to 3 dose, per day.

5. Coconut and Cucumber:
Mix equal quantities of coconut water and cucumber leaves juice to relieve excessive thirst and prevent dehydration.

6. Yoghurt:

Angina (Chest Pain)

Instruction

1. Garlic:
Add ½ teaspoon of garlic juice to a cup of hot water and drink it.

2. Ginger:
Add 1 tablespoon of grated ginger to a cup of hot water. Cover and steep for 5 minutes, then strain it and drink it.

3. Lemon:
Try to include one lemon in your food each day. You can squeeze it over salads or have it as fresh lemon water.

4. Basil Leaves:
Chew a few leaves of fresh basil leaves in the morning. If you do not get fresh leaves, then make a juice of basil, adding a few spoons or concentrated basil juice and warm water.

5. Grapes:
These are known to strengthen the heart. They help to reduce the risk of heart attack, angina pain and increase the quality of your breath.

Pneumonia

Instruction

1. Garlic
Mix one cup of milk in four cups of water and add one-half teaspoon of crushed garlic to it. Boil the solution until it reduces to one-fourth of the original amount. Drink it three times a day.

2. Peppermint Oil
Mix a few drops of peppermint oil with a carrier oil of your choice. Apply this mixture to your chest and back.

3. Ginger
Ginger is a well known remedy to treat most of the respiratory diseases. Taking home made ginger tea will improve cough and lung irritation. It's pretty effective in treating flu too.

4. Cayenne Pepper
Mix some lemon juice and a dash of cayenne pepper in an eight-ounce glass of water. Drink it a few times a day.

5. Practice Good Hygiene

Figure 7.2 *Doctor at Home* claims to be able to 'cure' critical diseases naturally.

Figure 7.2. Additionally, the app states that the user can be a home doctor, defined as "you are yourself a doctor". Herbal treatments and reference material cannot replace professional diagnosis or treatment. While the app has useful tips for treating simple symptoms and issues, such as coughing and dandruff, it also has claims for treating more serious cases such as stomach ulcers and cholera.

Pain Relief. These apps rely on providing exercises and remedies to address various types of muscular pains or migraines. While such apps can offer a catalogue of exercises that can address certain types of pain, they should be used in conjunction with medical advice. Apps which work in tandem with qualified pain coaches can be a convenient way to help manage pain remotely. However, for many pain relief apps, pain is addressed through virtual exercises with claims that they are "proven to ease pain", such as in app *Lower Back Pain and Sciatica Relief Exercises*. Note that the issue is not whether exercises are effective or not; rather the issue is that app descriptions do not suggest seeking professional medical advice prior to app installation. Additionally, certain pain exercises, when performed incorrectly or without supervision, can lead to further damage and pain in many cases [99].

7.7 Self Assessment

We have found 500 apps which emphasize the use of assessments and self-help, categorized as follows.

Mental Health. Mental health apps rely on self-assessments without a professional entity providing feedback. Note that there is a lack of direct scientific evidence found in descriptions of apps that claim to help with mental health or behavioral patterns [93]. Many mental health apps do not provide confirmation or verification that the app is indeed vouched for by professionals. For example, the self-help app *MoodSpace* is focused on depression and mental well-being. While the description claims that the app is "a well-being app driven by research", there is no evidence of any research or authoritative proof accessible to users prior to installation. As with prior examples, the app's description contains a disclaimer and an emphasis that users should seek medical advice, but the disclaimer is found at the very end of the description, increasing the chance to be ignored by users.

Symptom Tracking. Symptom tracking apps are based on user input (rather than physiological measurements as prior discussed) to determine possible diagnoses. These apps are useful for a cursory understanding of certain symptoms but should not be used for a diagnosis. Many of these apps are extremely popular, such as *Ada-check your health*, with over 5,000,000 installs and classified as a Class I Medical Device, meaning it is considered as a device with low risk to the user in the European Union. While Ada-check your health is an example of a well-regulated medical app, there are many apps that claim to perform similar functions but are not as well scrutinized or moderated by government or marketplace entities, such as the Disease Detector which claims to detect diseases in a few seconds.

Pregnancy Quizzes. These apps ask a series of questions and claim to determine whether the user shows early signs of pregnancy. While a collection of certain symptoms can help determine the likelihood of pregnancy, it can only be validated through an actual physical pregnancy test. As a result, the framing and naming of these apps are misleading. An example was the app *Real Pregnancy Test Quiz* – removed from Google Play during this research – which suggested that it was "an easy quiz for pregnancy. Just reply the quiz questions".

7.8 Conclusions

Medical mobile apps are understandably convenient and appealing to users. However, app quality and app description quality remain sorely lacking. These lacunae are particularly concerning in this (medical) domain because app reliability can directly affect/impact user safety and well-being. Our approach and study found that the functionality landscape of medical apps is broad and varied; however, the functionalities claimed in app descriptions are not entirely reliable. Our findings show a need for better regulation and scrutiny of medical apps in-app marketplaces to better protect users and their health.

CHAPTER 8

VERIFICATION FOR MEDICAL SCORES AND SCORE CALCULATOR APPS

We will now discuss issues regarding medical score calculator apps. In this chapter, we discuss and reveal inaccuracies found in medical score calculator apps and in their respective reference charts. Mobile medical score calculator apps are widely used among practitioners to make decisions regarding diagnosing and treating patients. Errors in score definition, input, or calculations can result in severe and potentially life-threatening situations. We address these issues via interval-based reference score verification as well as app verification and validation, as follows. We first introduce a model for checking the correctness of the reference scoring systems (score specification). Specifically, we reduce score correctness to partition checking (coverage and non-overlap) over score parameters' ranges.

8.1 Introduction

The adoption of mobile medical apps in a clinical setting is already strong.[1] While these apps are appealing by helping practitioners compute scores or dosages, app reliability has received little attention. Reliability is particularly important in acute care, where accuracy can decisively influence outcomes. For example, a 2021 study regarding emergency department personnel has revealed that 91.8% of those surveyed used medical apps on their devices during their shifts in the midst of heavy workloads and a stressful environment. Among the most used apps, 66.7% were medical scoring calculators [81]. Another study showed that 98% of acute care nurses used a smartphone in acute settings "to access information on medications, procedures, and diseases" [67].

[1] Clinical smartphone use among physicians: 70% and above as early as 2012 [110, 148].

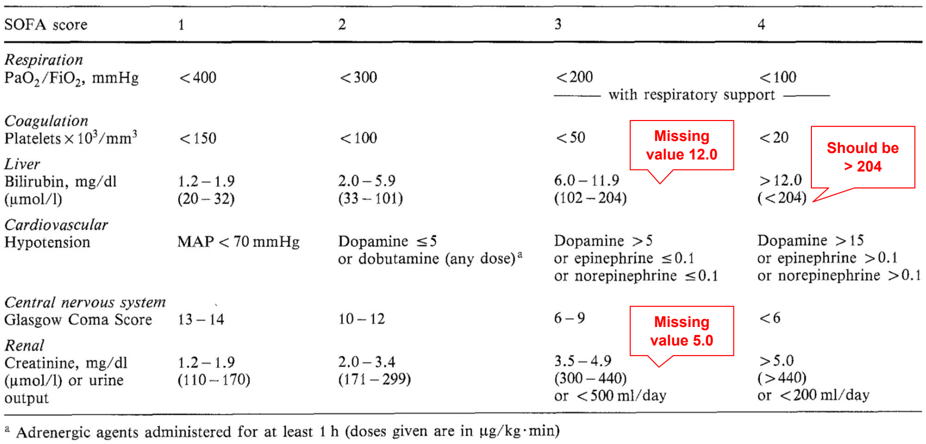
Our focus is the accuracy of medical score calculators, i.e., apps that compute a medical score based on supplied parameters. Such scores are ubiquitous in triage, the ICU, and determining the rate of rapid decline. For example, the Modified Early Warning Score (MEWS) determines whether a patient's state is likely to deteriorate quickly, potentially warranting ICU admission. The Sepsis-related Organ Failure Assessment (SOFA) is used in the ICU to determine the rate of organ dysfunction in ICU patients. Traditionally, scores were calculated manually from reference tables which map parameter values to several score components. These components are added to obtain an overall score, which is checked against an action threshold.

Reference tables are usually defined in medical research papers or regulatory documents. According to our observations, two main issues impact score calculator apps' correctness. First, the reference tables themselves can be inconsistent, which can lead to erroneous scores even when the score is computed manually. Second, the implementation of these scoring systems in apps can be incorrect, either due to developer errors in general or due to developer confusion induced by attempting to implement an inconsistently-defined score.

Incorrect scores can have dire consequences. For example, the Modified Early Warning Score (MEWS) [70] determines whether a patient should be moved to the ICU or not based on the score value being ≥ 4. Thus, when a score calculator produces an incorrectly low score, the severity of the condition is underestimated, resulting in misdiagnosis and other negative patient outcomes.

This area continues to be under-scrutinized and under-regulated. We are not aware of any prior attempts to verify reference scores themselves or to validate apps automatically.

Other prior efforts, whether targeted studies on apps implementing calculators for dosing parameters for opiate medications [75] or insulin [82], as well as medical app meta-studies [30], have revealed significantly different results across apps for the



SOFA score	1	2	3	4
Respiration PaO$_2$/FiO$_2$, mmHg	< 400	< 300	< 200	< 100
			—— with respiratory support ——	
Coagulation Platelets × 10^3/mm^3	< 150	< 100	< 50	< 20
Liver Bilirubin, mg/dl (µmol/l)	1.2 – 1.9 (20 – 32)	2.0 – 5.9 (33 – 101)	6.0 – 11.9 (102 – 204)	> 12.0 (< 204)
Cardiovascular Hypotension	MAP < 70 mmHg	Dopamine ≤ 5 or dobutamine (any dose)[a]	Dopamine > 5 or epinephrine ≤ 0.1 or norepinephrine ≤ 0.1	Dopamine > 15 or epinephrine > 0.1 or norepinephrine > 0.1
Central nervous system Glasgow Coma Score	13 – 14	10 – 12	6 – 9	< 6
Renal Creatinine, mg/dl (µmol/l) or urine output	1.2 – 1.9 (110 – 170)	2.0 – 3.4 (171 – 299)	3.5 – 4.9 (300 – 440) or < 500 ml/day	> 5.0 (> 440) or < 200 ml/day

[a] Adrenergic agents administered for at least 1 h (doses given are in µg/kg · min)

Missing value 12.0

Should be > 204

Missing value 5.0

Figure 8.1 SOFA Score (from Vincent et al. [147]).

same calculation, incorrect dosages, potential harmful recommendations, and a lack of medical professionals involved in app creation.

Hence there is an impetus for medical score calculators and medical apps in general to be highly scrutinized. In practice they are not, as they fall out of the scope of many regulatory bodies. For instance, in the US, the FTC (Federal Trade Commission) currently only regulates mobile medical apps that are used as a device or connect with a device, e.g., an insulin pump [69].

8.2 Motivation

We first define the key terms and concepts used throughout the chapter, and then provide a suite of examples – actual reference tables and app errors – to justify our approach.

8.2.1 Definitions

Reference Table. We use the term *reference table* for the table in the form it was first introduced in a medical research article or a regulatory agency document. For example, the Sequential Organ Failure Assessment (SOFA) score, shown in Figure 8.1 and discussed shortly, was introduced by Vincent et al. [147] in the *Intensive Care Medicine* research journal in 1996. The NEWS (National Early Warning Score), another score we consider, was introduced by the UK's National Health Service (NHS) in 2012 and later updated in 2017 to NEWS2 [23]. Score tables are structured as follows: most commonly, each cell in the table contains intervals for one physiological parameter, while the row or column header contains a numeric value, typically 0–4. For example, in SOFA's reference table (Figure 8.1) the third row shows intervals 1.2–1.9, 2.0–5.9, and so on, for parameter *Bilirubin*. The header row in the table shows numeric values, in SOFA's case 1 through 4, which correspond to individual scores for the intervals in that column. Occasionally, a table entry contains just a threshold value, e.g., *MAP* $<$ *70 mmHg* in SOFA's fourth row. Finally, a cell can contain intervals or thresholds for more than one parameter, e.g., *Dopamine* > 15 *or norepinephrine* > 0.1 in SOFA's fourth row. Next we describe score computation.

Score. A score is computed by adding the individual scores corresponding to each cell. For example, a patient with *Respiration=350*(score=1), *Coagulation=90* (score=2), *Liver=7.0* (score=3), *Central nervous system=13* (score=1), *Renal=1.5* (score=1) would have an overall SOFA score:

$$SOFA\, score = 1 + 2 + 3 + 1 + 1 = 8$$

The overall score value determines the course of action. In Table 8.2 (discussed at length later) we show action thresholds, e.g., "aggressive treatment if HEART score ≥ 7"; hence an accurate value is critical for patients' health outcomes.

As we will illustrate shortly, many errors, in reference tables themselves or the apps implementing the tables, stem from violations of the following two partition conditions: coverage (exhaustion), and non-overlap (disjointness). Coverage violations occur when specific interval values are not covered. For example, for intervals [0-3] and [4-6] over real numbers, any values between 3 and 4 are not covered, resulting in a violation. Non-overlap violations occur when there are overlapping

Table 8.1 Medical Scores Analyzed, the Year Scores Were Introduced, Errors Found, Score Ranges, and Action Thresholds

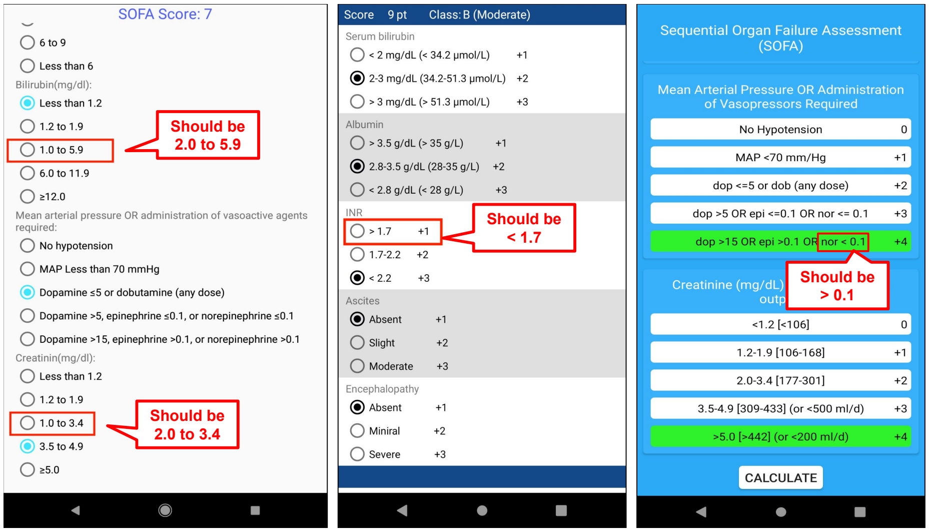
	Score Name	**Year**	**Errors**		**Range**	**Thresholds**
			Interval/Value Not Covered	**Overlap**		
Error Found	SOFA *(10,414 citations)*	1996	Bilirubin=12.0 Creatinine=5.0	Bilirubin ([102-204], [<204])	0-24	$\geq$ 11: "higher mortality rate" [66]
	HEART *(522 citations)*	2008		Age([$\leq$45], [45-65], [$\geq$ 65])	0-10	4-6: "cannot discharge; admit for clinical observation, noninvasive investigation" $\geq$ 7: "early aggressive treatment including invasive strategies" [126]
	Pulmonary Asthma Score *(150 citations)*	2002	Resp. rate (<6 yrs)=30 Resp. rate ($\geq$6 yrs)=20		0-9	9 "severe exacerbation" [127]
	APACHE II *(21,152 citations)*	1985	Age=44		0-71	25: "predicted mortality of 50%" $\geq$ 35 "predicted mortality of 80%" [47]
	RAPS "retold" *(357 citations)*	2004	Respiratory rate=5 Heart rate=39 Mean arterial press.=49		0-20	$\geq$ 7 "increased mortality" [109]
No error found	MEWS	2006			0-18	$\geq$ 4: "surgical team should be informed immediately" [70]
	NEWS	2012			0-18	Any value of 3 in a parameter: "urgent ward based response" 5 or 6: "key threshold for urgent response" $\geq$ 7: "urgent or emergency response" [49]
	NEWS2	2017			0-21	5 or 6: "patient should be monitored" $\geq$ 7: "urgently inform a clinician competent in the assessment of acutely ill patients" [108]
	Child Pugh	1973			0-15	8-10: "increased mortality" $\geq$ 11: "hepatic failure" [163]
	HAS-BLED	2010			0-9	$\geq$ 3 "high risk of bleeding" [97]
	CHA2DS2VASc	2010			0-9	$\geq$ 2 "high risk of stroke and thromboembolism" [96]
	Glasgow Coma Scale	1974			3-15	13-15: "mild neuroemergency " 9-12: "moderate neuroemergency" 6-8: "aggressive triage and prompt neurosurgical and critical care management" 3-5 "mortality is high and long-term neurological outcomes are generally poor" [33]

ranges: for intervals [0-3] and [3-5], 3 is found in both intervals, resulting in a non-overlap violation.

8.2.2 Error Source #1: Inconsistent Reference Table

Inconsistent definitions in reference tables are the most concerning kind of errors we found, because, unlike apps, tables are difficult to update or fix. Moreover, as our evaluation shows, an incorrect reference table is likely to lead to incorrect implementations in apps, because developers tend to implement tables *ad literam*. Finally, an inconsistent table will lead to an inconsistent GUI that confuses app users and invites score calculation errors.

We illustrate several such inconsistencies on the *SOFA Score* reference table. The SOFA score, introduced in 1996, predicts ICU mortality by evaluating the dysfunction of six systems by scoring each organ from 0, which is considered normal



SOFA Score: 7

○ 6 to 9
○ Less than 6

Bilirubin(mg/dl):
● Less than 1.2
○ 1.2 to 1.9
○ 1.0 to 5.9
○ 6.0 to 11.9
○ ≥12.0

Should be 2.0 to 5.9

Mean arterial pressure OR administration of vasoactive agents required:
○ No hypotension
○ MAP Less than 70 mmHg
● Dopamine ≤5 or dobutamine (any dose)
○ Dopamine >5, epinephrine ≤0.1, or norepinephrine ≤0.1
○ Dopamine >15, epinephrine >0.1, or norepinephrine >0.1

Creatinin(mg/dl):
○ Less than 1.2
○ 1.2 to 1.9
○ 1.0 to 3.4
● 3.5 to 4.9
○ ≥5.0

Should be 2.0 to 3.4

Score 9 pt Class: B (Moderate)

Serum bilirubin
○ < 2 mg/dL (< 34.2 µmol/L) +1
● 2-3 mg/dL (34.2-51.3 µmol/L) +2
○ > 3 mg/dL (> 51.3 µmol/L) +3

Albumin
○ > 3.5 g/dL (> 35 g/L) +1
● 2.8-3.5 g/dL (28-35 g/L) +2
○ < 2.8 g/dL (< 28 g/L) +3

INR
○ > 1.7 +1
○ 1.7-2.2 +2
● < 2.2 +3

Should be < 1.7

Ascites
● Absent +1
○ Slight +2
○ Moderate +3

Encephalopathy
● Absent +1
○ Miniral +2
○ Severe +3

Sequential Organ Failure Assessment (SOFA)

Mean Arterial Pressure OR Administration of Vasopressors Required

No Hypotension	0
MAP <70 mm/Hg	+1
dop <=5 or dob (any dose)	+2
dop >5 OR epi <=0.1 OR nor <= 0.1	+3
dop >15 OR epi >0.1 OR nor < 0.1	+4

Should be > 0.1

Creatinine (mg/dL) outp

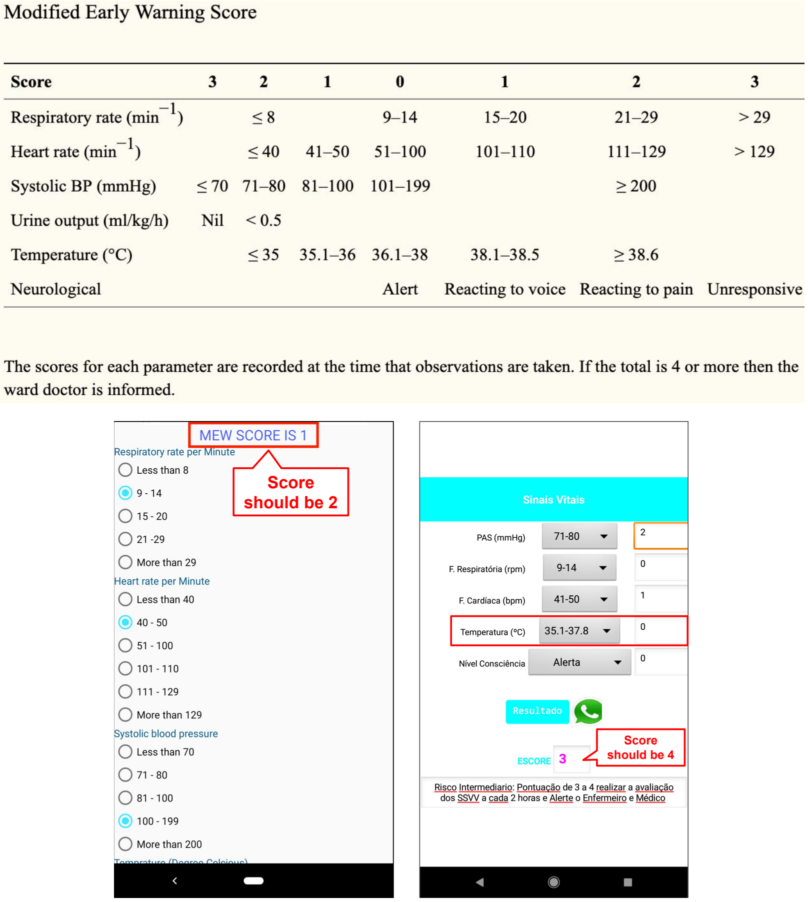
<1.2 [<106]	0
1.2-1.9 [106-168]	+1
2.0-3.4 [177-301]	+2
3.5-4.9 [309-433] (or <500 ml/d)	+3
>5.0 [>442] (or <200 ml/d)	+4

Figure 8.2 Inconsistent GUI errors in three apps: *Nursing Calculator* (left), *Child-Pugh Score* (center), *SOFA 1.2.0* (right).

functionality, to 4, the most abnormal [147]. Thus the highest possible score to obtain would be 24, indicating severe morbidity, and the lowest would be 0.

The reference table for the SOFA score is shown in Figure 8.1. Notice how for Liver (Bilirubin), the second-to-last interval is defined as 6.0–11.9. As the parameter is a real number, the actual interval specification is $[6.0, 12.0)$. That is, a value such as 11.95 would still be in the interval because only the first decimal is specified. The last interval for bilirubin is > 12.0. Hence the interval-based specification for these two entries is: $[6.0, 12.0)$ and $(12.0, max)$. *This squarely violates the coverage property of the partition, because value 12.0 is not covered by any interval.* The same issue is present for parameter Renal (Creatinine), where value 5.0 is not covered. It is unclear how developers are supposed to cope with this incorrect specification, e.g., the SOFA score of a patient with Bilirubin=12 and Creatinine=5 can be *off by as much as 2 points*, depending on how the table is interpreted.

Finally, when bilirubin is specified in $\mu mol/l$, the table's last column shows '(< 204)' which is incorrect: the entry should be '(> 204)' (note how values < 204 are already covered in the preceding intervals). If the developer implements the table



Modified Early Warning Score

Score	3	2	1	0	1	2	3
Respiratory rate (min^{-1})		≤ 8		9–14	15–20	21–29	> 29
Heart rate (min^{-1})		≤ 40	41–50	51–100	101–110	111–129	> 129
Systolic BP (mmHg)	≤ 70	71–80	81–100	101–199		≥ 200	
Urine output (ml/kg/h)	Nil	< 0.5					
Temperature (°C)		≤ 35	35.1–36	36.1–38	38.1–38.5	≥ 38.6	
Neurological				Alert	Reacting to voice	Reacting to pain	Unresponsive

The scores for each parameter are recorded at the time that observations are taken. If the total is 4 or more then the ward doctor is informed.

MEW SCORE IS 1

Score should be 2

Respiratory rate per Minute

○ Less than 8
● 9 - 14
○ 15 - 20
○ 21 -29
○ More than 29

Heart rate per Minute

○ Less than 40
● 40 - 50
○ 51 - 100
○ 101 - 110
○ 111 - 129
○ More than 129

Systolic blood pressure

○ Less than 70
○ 71 - 80
○ 81 - 100
● 100 - 199
○ More than 200

Temprature (Degree Celcious)

Sinais Vitais

PAS (mmHg)	71-80 ▼	2
F. Respiratória (rpm)	9-14 ▼	0
F. Cardíaca (bpm)	41-50 ▼	1
Temperatura (ºC)	35.1-37.8 ▼	0
Nível Consciência	Alerta ▼	0

Resultado

ESCORE 3

Score should be 4

Risco Intermediario: Pontuação de 3 a 4 realizar a avaliação dos SSVV a cada 2 horas e Alerte o Enfermeiro e Médico

Figure 8.3 Top: MEWS reference table. Bottom: *Nursing Calculator* incorrect score (left); *MEWS Brasil* incorrect scores for *Temperature* and overall (right).

ad literam and offers '(< 204)' as a GUI option, the SOFA score of a patient can be *off by as much as 3 points.* Note that even a off-by-one error can affect patients' condition classification between "patient should be monitored" and "urgently inform a clinician." Section 8.4.2 discusses these issues at length.

8.2.3 Error Source #2: Inconsistent GUI

We now turn to the first kind of implementation errors, where the GUI is inconsistent. Our approach detects two kinds of errors. In the first kind, the user can input the same parameter value into *two different GUI boxes*, which impacts the score. In the second kind, there is no input box for a certain value. These embody violations of the coverage and non-overlap conditions, respectively. We discuss how we check GUIs for such errors automatically via dynamic analysis and constraint solving.

Example 1: SOFA score in app *Nursing Calculator*. The app *Nursing Calculator*[2], with over 50,000 installs, provides a variety of medical calculators, including the SOFA score. The app's GUI has two inconsistency errors (first kind), as highlighted in Figure 8.2 (left), and described next. The option for Bilirubin shows a range of 1.0–5.9 when it is supposed to be 2.0–5.9. Moreover, for Creatinine the range in the app is 1.0–3.4, when it should be 2.0–3.4. Due to these errors, a patient's score can be *off by as much as 2 points*.

Example 2: Child-Pugh score in app *Child-Pugh Score*. The Child-Pugh score is generally used to assess the potential for liver diseases, mainly cirrhosis. The app *Child-Pugh Score*[3] has an inconsistency error (first kind) regarding values for INR, as highlighted in Figure 8.2 (center): the first option should be '< 1.7' instead of '> 1.7'. Due to this error, a patient's score can be *off by as much as 2 points*.

Example 3: SOFA score in app *SOFA 1.2.0*. This app exemplifies the second kind of error. The app, *SOFA 1.2.0*,[4] removed from Google Play in the course of our research, exhibited a GUI inconsistency error as highlighted in Figure 8.2 (right). Note that the reference table's last column in the *Cardiovascular* row specifies the score for . . . *norepinephrine* > 0.1; the app however incorrectly lists '*norepinephrine* < 0.1'. The app offers no option where the provider can indicate the *norepinephrine* > 0.1 condition, potentially altering the score by 1 point.

[2] `https://play.google.com/store/apps/details?id=com.niya.lijo.nursingcalculators`

[3] `https://play.google.com/store/apps/details?id=br.child_pugh`

[4] `https://apksos.com/app/com.varendrasoft.sofascore`

8.2.4 Error Source #3: Incorrect Score Calculation

Even with a consistent table and consistent GUI, apps can still be prone to errors in score calculation, such as an instance where the table the score is 4, but the app displays 6. These calculation errors are silent and particularly pernicious since the user does not have any indication that the calculation has gone awry.

We now present several examples based on the Modified Early Warning Score (MEWS), which is used by professionals to determine whether or not a surgical in-patient requires intensive care [70]. The reference table, which we verified as consistent, is shown in Figure 8.3 (top).

When the app *Nursing Calculator* starts, parameter values are in their default settings, where individual scores are 0 corresponding to a MEWS score of 0. After the user changes the Heart rate to 40, the output score is 1 instead of the expected value of 2 (screenshot in Figure 8.3 bottom-left). Note that a higher MEWS value indicates a more severe situation.

Another example is found in *MEWS Brasil* [25], shown in Figure 8.3 bottom-right. Suppose the user inputs Heart rate between 41–50 and BP between 41–50; cumulatively, the score is 3. The error manifests when the patient temperature is in the interval 35.1–36; per the table, the individual temperature score value is 1. In the app however, the value is 0, hence the displayed overall MEWS score, 3, is incorrect (the correct value is 4). This error is particularly problematic, because 4 is a threshold value: "[at $\geq$ 4] surgical team should be informed immediately" [70].

These incorrect implementations can potentially lead to a big difference in medical decision-making and outcomes, making it crucial to verify these scoring systems. The target audience of these scoring apps is also concerning. There is evidence that users of medical calculator apps are clinicians and nurses, especially *inexperienced and younger doctors*, according to a 2021 study [81]. Another study among surgeons from 2015 found that "Junior doctors were more likely to use medical

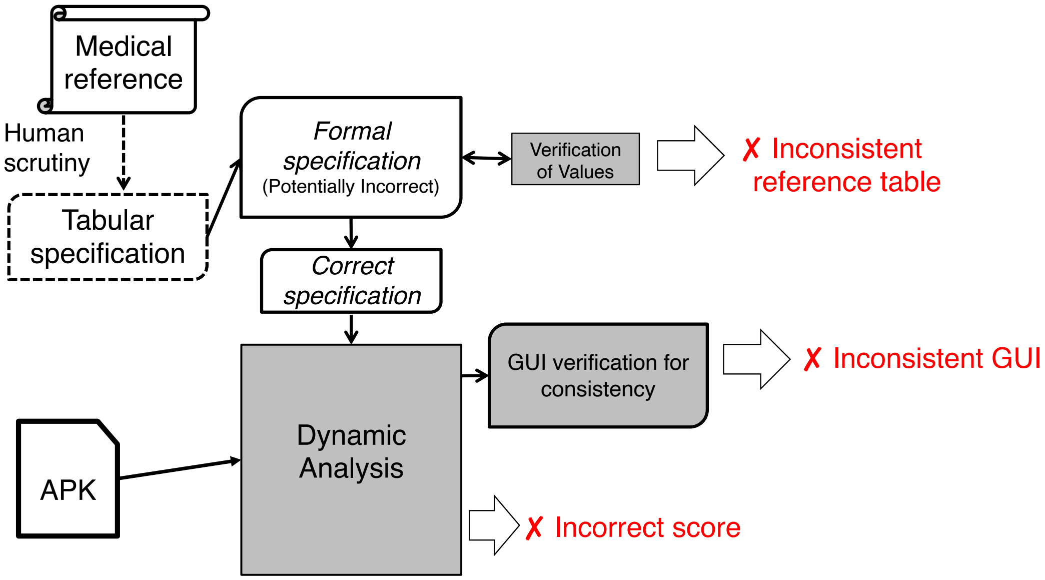


Figure 8.4 Overview of our approach and toolchain.

apps over their senior colleagues (p = 0.001) as well as access the Internet on their smartphone for medical information (p <0.001)" [112].

8.3 Approach

We now summarize our approach to finding errors in reference tables, app GUIs, and app score calculations. In the first stage for each scoring system, we extract a *formal specification* for the reference table, which our toolchain then checks for consistency. We then fix the inconsistency found in the reference table before using it as a reference. Next, for a given APK implementing that score, our toolchain first performs a dynamic analysis[5] to extract a *GUI specification*. The GUI specification is (a) validated against the correct reference table specification, and (b) verified for consistency. Finally the app's output score is verified against the reference score for that input parameter combination.

[5] The consistency check for formal specifications and the dynamic analysis are not contributions of this work.

8.4 Verifying Reference Scores

We evaluated our approach on 12 medical scores. We focused on scores used in critical settings, where errors have serious implications. The scores, ranges, potential errors and action thresholds are shown in Table 8.2. We first discuss scores' nature, argue why score accuracy is critical, and then present the errors we found.

8.4.1 Reference Scores

We only provide details and citations numbers in (Table 8.2, first column) for those scores that contain errors.

The *SOFA* (Sequential Organ Failure Assessment) score predicts the mortality rate of an ICU patient based on the functionality of six organ systems. The score is updated and calculated every 24 hours until the patient is discharged [147].

The *HEART* (History, EKG, Age, Risk Factors, Troponin) score is used to predict the risks of a major cardiac event while taking into account risk factors from a patient's history or age and other parameters [126].

The *RAPS* (Rapid Acute Physiology Score) is used to predict mortality of a patient in a critical care transport [121].

The *Pulmonary Asthma Score* was developed to simplify a measure to determine asthma severity in children [127].

APACHE II (Acute Physiology and Chronic Health Enquiry II) is used to provide a general measure of disease severity while taking into account current measurements, age, and health history [89].

The remaining scores do not contain errors (though apps implementing the scores do); the scores' domains are as follows: identifying the severity of patients' conditions in critical care (*MEWS* [70], *NEWS* [49], *NEWS2* [23]); chronic liver disease severity (*Child-Pugh* [163]); risk of bleeding (*HAS-BLED* [97]); stroke risk (*CHA2DS2VASc* [96]); and severity of a brain injury (*Glasgow Coma Scale* [134]).

Table 8.2 Medical Scores Analyzed, the Year Scores Were Introduced, Errors Found, Score Ranges, and Action Thresholds

	Score Name	**Year**	**Errors**		**Range**	**Thresholds**
			Interval/Value Not Covered	**Overlap**		
Error Found	SOFA *(10,414 citations)*	1996	Bilirubin=12.0 Creatinine=5.0	Bilirubin ([102-204], [<204])	0-24	$\geq$ 11: "higher mortality rate" [66]
	HEART *(522 citations)*	2008		Age([$\leq$45], [45-65], [$\geq$ 65])	0-10	4-6: "cannot discharge; admit for clinical observation, noninvasive investigation" $\geq$ 7: "early aggressive treatment including invasive strategies" [126]
	Pulmonary Asthma Score *(150 citations)*	2002	Resp. rate (<6 yrs)=30 Resp. rate ($\geq$6 yrs)=20		0-9	9 "severe exacerbation" [127]
	APACHE II *(21,152 citations)*	1985	Age=44		0-71	25: "predicted mortality of 50%" $\geq$ 35 "predicted mortality of 80%" [47]
	RAPS "retold" *(357 citations)*	2004	Respiratory rate=5 Heart rate=39 Mean arterial press.=49		0-20	$\geq$ 7 "increased mortality" [109]
No error found	MEWS	2006			0-18	$\geq$ 4: "surgical team should be informed immediately" [70]
	NEWS	2012			0-18	Any value of 3 in a parameter: "urgent ward based response" 5 or 6: "key threshold for urgent response" $\geq$ 7: "urgent or emergency response" [49]
	NEWS2	2017			0-21	5 or 6: "patient should be monitored" $\geq$ 7: "urgently inform a clinician competent in the assessment of acutely ill patients" [108]
	Child Pugh	1973			0-15	8-10: "increased mortality" $\geq$ 11: "hepatic failure" [163]
	HAS-BLED	2010			0-9	$\geq$ 3 "high risk of bleeding" [97]
	CHA2DS2VASc	2010			0-9	$\geq$ 2 "high risk of stroke and thromboembolism" [96]
	Glasgow Coma Scale	1974			3-15	13-15: "mild neuroemergency " 9-12: "moderate neuroemergency" 6-8: "aggressive triage and prompt neurosurgical and critical care management" 3-5 "mortality is high and long-term neurological outcomes are generally poor" [33]

8.4.2 Why is Score Accuracy Critical?

We chose these scores because they capture critical conditions where action is urgently needed. Errors in the app-calculated scores can result in under-estimating the real score, which potentially means that time-sensitive life-saving interventions will not be taken. Conversely, errors that result in the app over-estimating the real score might lead to an overly aggressive, disproportionate intervention, as well as unnecessary use of resources (personnel and ICU beds). For each score, Table 8.2's second-to-last and last columns show the range of possible values and threshold values, respectively; the third and fourth columns show errors (if any) and will be discussed. The threshold column is particularly revealing, as it indicates the score value(s) at which a more aggressive intervention is warranted, or values where the prognosis turns dim. For example, for the HEART score, a patient who "scores" ≤ 3 can be discharged; a patient who scores 4–6 would be admitted for noninvasive investigation; whereas a patient who scores ≥ 7 will receive "early aggressive treatment including invasive strategies." Hence a score calculation error at or around the threshold value is particularly concerning.

8.4.3 Inconsistent Reference Table

We found errors in the original reference tables for 4 of the 11 scores. We also found errors in one score as defined in follow-up work to the original reference table; these errors are shown in the top part of Table 8.2.

For SOFA, the partition condition (1) coverage, is violated for *Bilirubin=12.0* and *Creatinine=5.0*; these values do not appear in the table though values lower or higher do appear in (Figure 8.1). The second issue for SOFA was a violation of the partition condition (2) non-overlap, where multiple table entries satisfy *Bilirubin* $<$ *204*. While the latter issue might be alleviated if an app/medical system does not use the μmol/l units, it is unclear how an app developer is supposed to deal with the former issue: should the 12.0 and 5.0 values be included into the left or right cells in the table?

The HEART score's reference table (relevant excerpt shown in Figure 8.5a) violates the non-overlap condition at two points: *Age=45* and *Age=65*; a possible resolution for app developers is to change the 45-65 interval to 46-64.

The Pulmonary Asthma Score's reference table (relevant excerpt in Figure 8.5b) violates coverage at two points: *Respiratory Rate=30* and *Respiratory Rate=20*; it is unclear how an app developer is supposed to cope with these, and whether the scores for those values should be 0 or 1.

The APACHE II reference table [89] violates the coverage condition for *Age=44*. This table would be particularly challenging to verify manually as it has 117 entries (13 rows by 9 columns).

The “RAPS retold” score was an interesting find. Note that the original RAPS score, introduced by Rhee et al. [121], does not violate the partition conditions. A new score, REMS, was introduced by Olsson et al. [109] to improve upon RAPS; the paper presents both scores, but the “retold” RAPS table (Figure 8.5d) has three coverage violations, as shown in Table 8.2.

8.5 Verifying Apps Score Implementations

We have evaluated our approach on a dataset of 90 apps. The selection processes are explained next, followed by discussing the errors we found, effectiveness and efficiency.

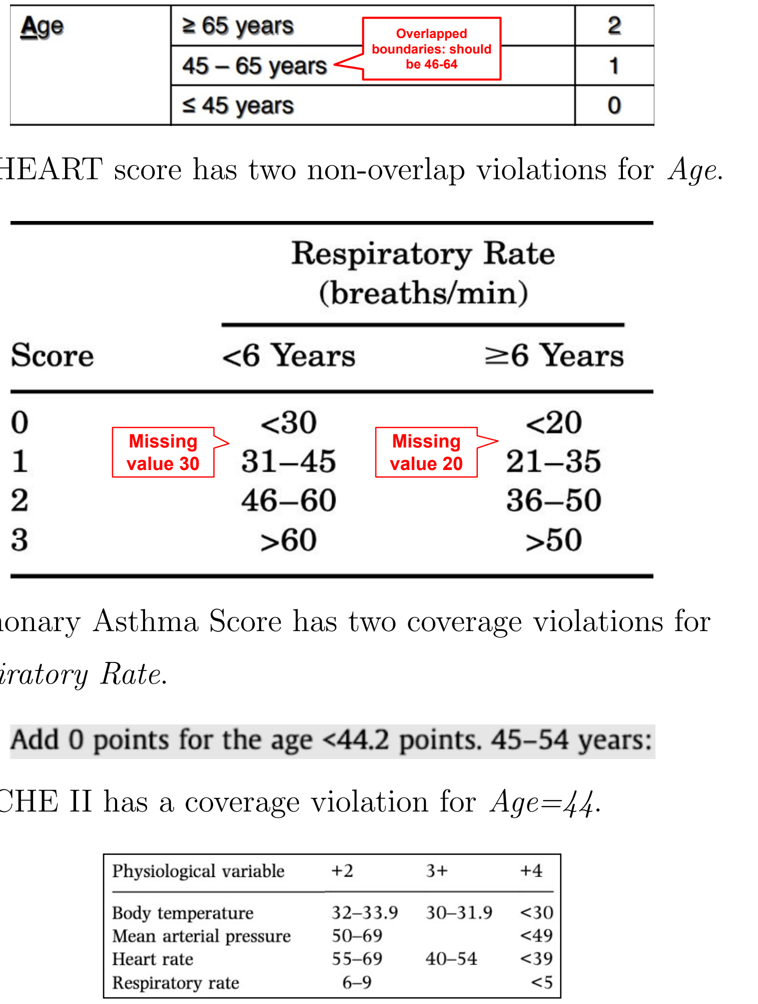
8.5.1 App Dataset

We selected our apps from the Medical category on Google Play. We scraped 3,762 apps and their descriptions from Google Play; using ranked retrieval, we identified 556 apps classified as any kind of medical calculator. We then focused on apps which computed one or more among the 12 scores we verified, resulting in a total of 90 apps.

8.5.2 App Errors: Inconsistent GUI

We now present our findings: coverage violations and non-overlap violations.

Coverage Violations. Table 8.3 shows the results; we found 23 coverage errors in 11 apps. In the first column we show the official app name on Google Play, in the second column we show the affected score calculator, and in the third column we show values or ranges that are not covered. Interestingly, though somewhat predictably, the “original sin” in the SOFA reference table (no coverage for *Bilirubin=12.0* and *Creatinine=5.0*) leads to non-coverage issues for the same parameter values in four apps.



Age	≥ 65 years	2
	45 – 65 years	1
	≤ 45 years	0

Overlapped boundaries: should be 46-64

(a) The HEART score has two non-overlap violations for *Age*.

Score	Respiratory Rate (breaths/min) <6 Years	 ≥6 Years
0	<30	<20
1	31–45	21–35
2	46–60	36–50
3	>60	>50

Missing value 30
Missing value 20

(b) Pulmonary Asthma Score has two coverage violations for *Respiratory Rate*.

Add 0 points for the age <44.2 points. 45–54 years:

(c) APACHE II has a coverage violation for *Age=44*.

Physiological variable	+2	3+	+4
Body temperature	32–33.9	30–31.9	<30
Mean arterial pressure	50–69		<49
Heart rate	55–69	40–54	<39
Respiratory rate	6–9		<5

(d) RAPS "retold" has three coverage violations for *Respiratory rate, Heart rate, and Mean arterial pressure.*

Figure 8.5 Reference tables with no straightforward fixes.

Non-overlap Violations. Table 8.4 shows the results: we found 18 coverage errors in 8 apps. The parameters with overlapping ranges are shown in the third column. In this case only one error, in the app *HEART Score*, appears attributable to the error in the original HEART reference table (Table 8.2).

8.5.3 App Errors: Incorrect Score Calculations

Table 8.5 shows errors in score calculation: we found 16 calculation errors in 16 apps. The *Calculation Errors* grouped columns show the parameter values for which

Table 8.3 Inconsistent GUI: Coverage Violations

App Name	Score	Parameter value(s)
Sepsis Clinical Guide	APACHE II	Hct(%) =60
Child Pugh Calculator	Child Pugh	INR =1.7
Child-Pugh Score (Blue Rock)	Child Pugh	INR < 1.7, INR > 2.2
HAS-BLED Score	HAS-BLED	Age=65

Nursing Calculator	MEWS	Systolic BP=70
		Resp=8
		Temp=35.0

Quick EM	SOFA	PaO2 ≥ 400 Dopamine=5
Nursing	SOFA	Bilirubin=12.0 Creatinine (mg/dl)=5.0
SOFA Score fn(widebitsbd)	SOFA	GCS=15 PaO2 ≥ 400 Platelets ≥ 150 Creatinine (mg/dl) <1.2

SOFA Score (Blue Rock)	SOFA	Bilirubin=12.0
		Creatinine (mg/dl)=5.0
		Dopamine=5

Merck Manual Professional	SOFA	Bilirubin=12.0 Creatinine (mg/dl)=5.0
SOFA	SOFA	Bilirubin=12.0 Creatinine (mg/dl)=5.0

the errors manifest, and the app value vs. reference score value. We make several observations. First, app errors lead to both under-estimating the true score (e.g., apps *Atrial fibrillation risk calc*, *MEWS*, *Nursing Calculator*-MEWS) and over-estimating the true score (e.g., apps *Sepsis3* or *MediCalc*). Both of these error types are problematic due to potential under-treating and over-treating patients respectively. Second, as the last column indicates, certain errors "straddle" the threshold, which, as discussed previously, can put the patient in a different class.

We have reached out to erroneous apps' developers. So far, 6 apps (listed in Table 8.6) have been fixed and updated.

8.6 Conclusions

Mobile health apps are increasingly utilized in acute care settings, and mobile app developers (including developers who are not medically qualified) are eager to capitalize on the surging demand for mobile health apps. Though errors in

Table 8.4 Inconsistent GUI: Non-Overlap Violations

App Name	Score	Overlapping Ranges
Child-Pugh Score	Child-Pugh	INR: [>1.7], [1.7-2.2]
HEART Score	HEART	Age: [$\leq$45], [45-65], [$\geq$ 65]
REBELEM	HEART	Age: [$\leq$45], [45-65], [$\geq$ 65]
MEWS	MEWS	Heart Rate: [51-101], [101-111] Respiratory rate: [15-21], [21-30] Systolic BP: [71-81], [81-101] Temperature: [$\leq$35], [35-38.5], [$\geq$38.5]
Nursing Calculator	SOFA	Bilirubin: [1.2-1.9], [1.0-5.9] Creatinine (mg/dl): [1.2-1.9] [1.0-3.4] Platelets: [$\geq$ 150], [100-150]
SOFA Score (Blue Rock)	SOFA	Creatinine (μmol/L): [110], [110-170],[300-440],[440]
SOFA	SOFA	Norepinephrine: [$\leq$0.1], [<0.1]
Sepsis Clinical Guide	SOFA	Creatinine (mg/dl): [$\leq$1.2], [1.2-1.9] Platelets: [$\geq$ 150], [$\leq$ 150]

Table 8.5 Calculation Errors in Apps

App Name	Score	Calculation Errors			Meets or Exceeds Threshold
		Parameter Option	App Score	Ref. Table Score	
Atrial fibrillation risk calc	CHA2DS2VASc	Age=75	1	2	N
Child-Pugh Score (KSoft Apps)	Child Pugh	INR=2.3	11	10	Y
Child-Pugh Score (Blue Rock)	Child Pugh	INR=2.3	11	10	Y
Child-Pugh Score Calculator - Liver Disease	Child Pugh	INR=2.3	11	10	Y
Nursing Calculator	MEWS	Heart rate=40	3	4	N
MEWS	MEWS	Temperature=[35.1-36]	3	4	N
Nursing Calculator	SOFA	Platelets=150	12	11	Y
Sepsis 3	SOFA	Dopamine=5	12	11	Y
Nursing	SOFA	PaO2=300	12	11	Y
MediCalc	SOFA	Dopamine=5	12	11	Y
SOFA Score(widebitsbd)	SOFA	Creatinine (μmol/L)=106	12	11	Y
Merck Manual Professional	SOFA	PaO2=300	12	11	Y
SOFA	SOFA	Creatinine(μmol/L)=106	12	11	Y
SOFA - Sepsis-related Organ Failure Assessment	SOFA	Dopamine=5	12	11	Y
Sepsis Clinical Guide	SOFA	Bilirubin=1.2	12	11	Y
Sepsis SOFA Calculator	SOFA	Platelets=20	4	3	Y

medical scores and score calculator apps can have severe negative consequences, such scores and apps are subject to little scrutiny. We have uncovered errors in long-standing medical reference articles. We found that incorrect specifications translate to incorrect app implementations and that even correct specifications can be implemented incorrectly, affecting the resulting scores. Our findings indicate a need for tighter scrutiny of reference scores themselves, as well as the apps implementing these scores.

Table 8.6 Apps Fixed Thanks to Our Reporting

App Name	**Package Name**	**#Installs**
Nursing	pe.com.codespace.nurse	100,000
MEWS Brasil	appinventor.ai_blinkeado.InformaticasaudeMEWS	500
Atrial fibrillation risk calc	com.gumptionmultimedia.atrialfibrillationriskscore	5,000
Nursing Calculator	com.niya.lijo.nursingcalculators	50,000
Sepsis Clinical Guide	app.escavo.sepsis	100,000
SOFA-Sepsis-related Organ Failure Assessment	gumptionmultimedia.com.sofascore	1,000

CHAPTER 9

REGULATING MEDICAL APPS

Supplementing the previous chapters' discussions of medical applications and uses, this chapter explains the regulations set in place to protect users. Medical mobile health apps provide a plethora of benefits, from helping patients keep track of their health, to allowing professionals to reference guidelines and access information conveniently. We break down the regulations of three medical entities within the US. We also classify whether medical apps should be regulated by the FTC and find a total of 192 potential apps that may require further scrutiny.

9.1 Introduction

Medical apps perform critical functions that involve patient data and other sensitive information. Overall, users of medical apps assume that apps are 'certified' and trustworthy when making medical decisions. A recent literature review from 2019 reveals that many consumer health apps have false content, poor design, and bad functionality [29]. Moreover, with the growing accessibility of mobile medical apps, there have been many cases of medical apps ultimately hurting the user. For example, an assessment of insulin dose apps revealed that the majority of such apps were inaccurate, leading to overdoses and deleterious mismanagement of glucose levels [83]. The question that arises is how many applications are federally approved and safe for use. The vast nature of mobile platforms makes it impossible to strictly regulate applications. Thus, analyzing federal guidelines regarding medical apps and determining whether applications are compliant towards federal mandates is necessary. While mobile health apps are used globally, there are many regulatory agencies and jurisdictions to keep track of. In an effort to provide a thorough analysis, we focus our research on US-based entities, namely the FDA and FTC, along with

HIPAA laws. After we define their bylaws and their rules overall, we will then check to see if applications fall within their scrutiny. By analyzing and classifying app metadata, mainly app descriptions and features, based on a subset of known regulated applications, we hope to reveal the vast number of applications that are not regulated but should be.

9.2 Regulatory Frameworks

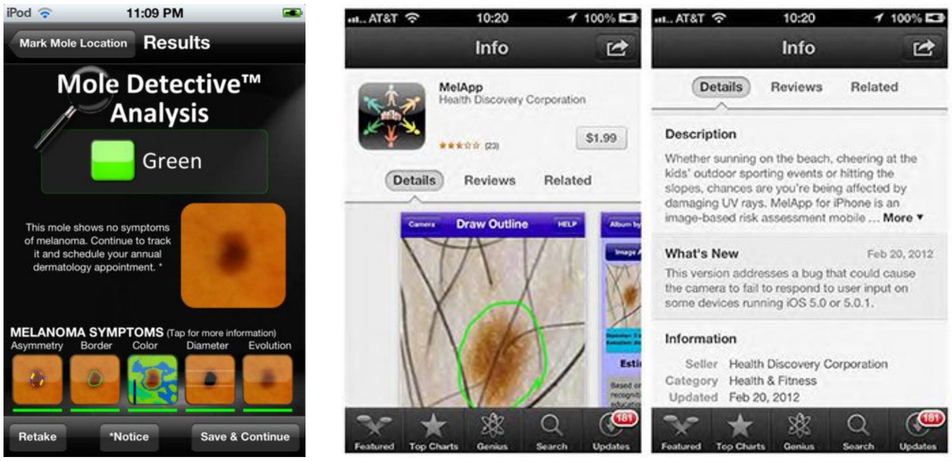
We begin by introducing and defining the regulatory frameworks that will be the center of this study.

9.2.1 FDA

The FDA, or Food and Drug Administration, is a US federal agency specialized in protecting the public from health hazards. The FDA approve and regulate medical devices; however, they also have jurisdiction over medical apps. Generally, their regulatory oversight focuses more on software functionality which directly impacts users. Outlined in the Policy for Device Software Functions and Mobile Medical Applications Guidance, which was first issued in 2013, then later updated in 2015, 2019, and 2022, the subset of software functions that are scrutinized are apps that turn the phone to a medical device [27]. These apps are all regulated under the Federal Food, Drug, and Cosmetic (FD&C) Act:

- Apps that connect to external medical devices, such as insulin pumps or glucometers, that are able to control and analyze data
- Apps connecting to hearing aids and are able to calibrate them
- Apps that transform the mobile device into a medical device via sensors or attachments, or by functionalities that are similar to these devices
- Apps that provide patient related analyses and output for health professionals and non professions, and are used for diagnosis and treatment (e.g., risk and dosage calculators)

The FDA, however does NOT regulate apps that do the following:



This mole shows no symptoms of melanoma. Continue to track it and schedule your annual dermatology appointment. *

Whether sunning on the beach, cheering at the kids' outdoor sporting events or hitting the slopes, chances are you're being affected by damaging UV rays. MelApp for iPhone is an image-based risk assessment mobile ... More ▾

This version addresses a bug that could cause the camera to fail to respond to user input on some devices running iOS 5.0 or 5.0.1.

Seller: Health Discovery Corporation
Category: Health & Fitness
Updated: Feb 20, 2012

Figure 9.1 *Mole Detective* (left) and *MelApp*(right).

- Help users manage diseases without specific treatment
- Remind users to take medications
- Coach users with healthy life tips such as nutrition and weight
- Automate clinical calculations and basic tasks for health professionals
- Uses the camera to document or send images in consultations
- Provides reference material or information to professionals and students

Overall, if an app is FDA-approved, the developers themselves initiated the process of getting it approved, as opposed to the FDA's monitoring of new apps before publication.

9.2.2 FTC

The Federal Trade Commission (FTC) enforces laws aimed to protect consumers from fraud, unfair practices, and deception. Regarding medical apps, the FTC is concerned with user privacy and security. There are two laws that medical apps must comply with. The first, the FTC Act, protects consumers from deceptive or misleading claims regarding app functionality and harm to the user. Additionally, the FTC enforces the Breach Notification Rule which requires developers to notify users and the FTC if there is a breach of unsecured personal health information. In other words, apps which shared personal health information without user authorization are penalized. Additionally, failure to comply results in a fine of $43,792 a day [69].

To aid developers, the FTC provides an interactive tool to better understand which laws and regulations comply with their application [69].

In 2015, the FTC filed a complaint against two melanoma detecting applications, *MelApp* and *Mole Detective*. The apps claimed to detect melanoma via a user provided photograph. The FTC stated that developers "must have scientific evidence to support any health or disease claims that they make for their apps" [21]. As a result, both apps were fined and removed from their respective app marketplaces.

Another example includes the female health tracking app, Flo, where the FTC filed a complaint against the app for exposing sensitive user data to third party apps without user consent [9].

9.2.3 HIPAA

The US Department of Health and Human Services (HHS) enforces legal rights via HIPAA (Health Insurance Portability and Accountability Act) which protects sensitive private health data. Many mobile medical apps acquire various types of user input related to their health. However not all user data is deemed vital or sensitive. PHI is information that can uniquely identify an individual. There are 18 specific PHI identifiers that HIPAA prioritizes that deal with patient info, such as name, date of birth, dates of treatments, account numbers and other information that would uniquely identify the patient [78].

Regarding how PHIs are handled by medical apps, HIPAA details a variety of scenarios explaining when an app would be under their jurisdiction [79].

Covered entities (e.g., health plans, providers) must comply with HIPAA Rules regarding PHIs of patients. Business associates who create apps for covered entities also are compliant. Apps that are subject to HIPAA rules are apps that involve or are associated with covered entities (e.g., telemedicine, electronic health record apps).

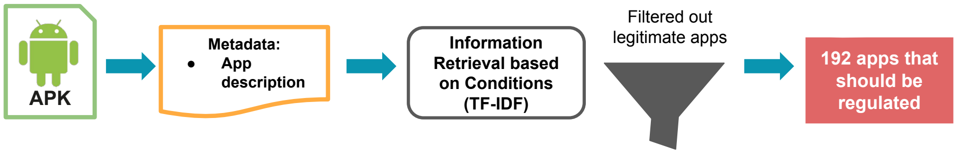

APK
Metadata:
• App description
Information Retrieval based on Conditions (TF-IDF)
Filtered out legitimate apps
192 apps that should be regulated

Figure 9.2 Methodology.

Even if an app does collect protected health information, as long as it is for personal use and will not be transmitted, it does not need to be cleared by HIPAA.

9.3 Challenges

There are many challenges involved in determining whether an app should be regulated. Firstly, app usage of sensors on its own and looking at app permissions are not reliable metrics for functionality and regulation; many apps suffer from having more permissions than intended.

Additionally, searching for PHI terms in an app's XML can also be misleading, as many apps require basic input data such as name, date of birth, and an email, all of which are PHIs according to HIPAA laws. Yet, they are also used in unregulated non-medical apps as well, which makes the case for regulation harder to consider.

Due to these challenges, we decided to focus on FTC regulations aimed to protect users from deception and fraud. Additionally, there are clear examples of apps that have violated these regulations, making it easier to observe patterns which can cause the FTC to act in response.

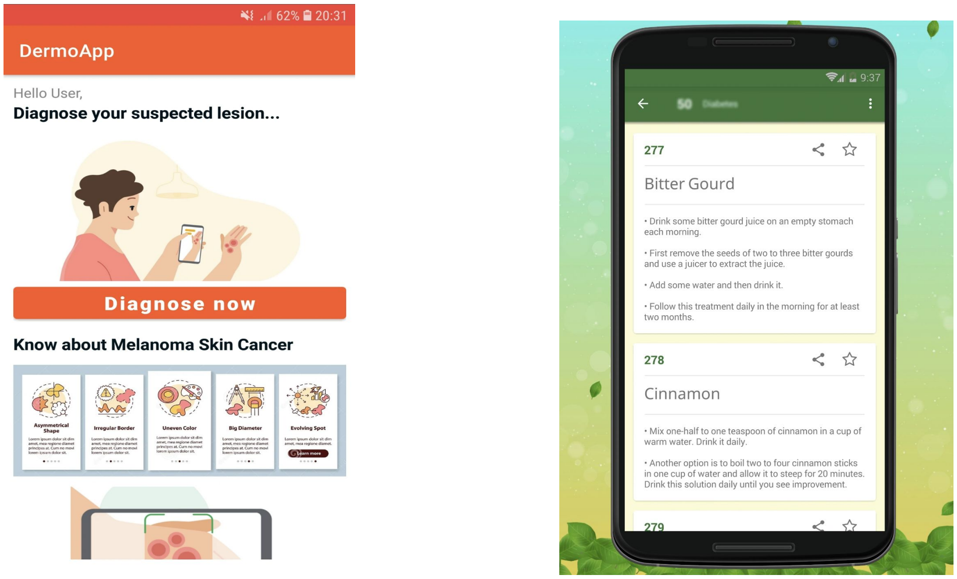
9.4 Results

Focusing on app descriptions allows us to observe how users can be misled and deceived. We begin by curating a list of keywords containing terms, such as 'treatment', 'cure', 'diagnosis', and medical conditions to utilize in a TF-IDF analysis of descriptions. We used a dataset of 1290 medical apps already kknown to have dubious claims. As a result, we identify a grouping of 347 apps that should be scrutinized further by the FTC for not having any scientific basis or confirmation. This grouping is achieved by apps that had a frequency of 6 or above, as these scores yielded the most relevant results. We considered apps that contained disclaimers or were recommended by medical professionals as false positives in our grouping. These apps, while not reliable or a replacement for a medical provider, are not inherently deceptive in practice as they do warn the user within the description. Thus, our false positive rate was 44.7%, with 155 apps. We did not have any false negatives in our analysis. Accordingly, there were 192 apps that we believe should be regulated.

We noticed certain patterns among the group of apps that should be FTC regulated or moved to a different category on Google Play. First, there are certain apps that contain the term "treatment" in their name, while not actually able to provide treatment via the mobile device. For example, the app *All Stomach Disease & Treatment* is a dictionary of various intestinal diseases. While it seems mostly harmless, the app states that "This clinical diagnosis and treatment book can help prevent brain problems by providing you with info about the brain eater stomach/intestinal worms. However, if it's late to prevent you just need to visit your doctor for treatment"[1]. We also found skin cancer detecting apps, like *MelApp* and *Mole Detective*, which were fined by the FTC and removed, that do not have disclaimers on its usage and offer a noninvasive detection method. The app, *DermoApp: Skincancer detection*, claims to have an accuracy rate of 96.8% which is not verified by any reputable medical source.

However, perhaps the most egregious cases of possible FTC violations are apps advertising various ancient, herbal, and religious treatments claiming to cure diseases. One such example is *Home Remedies Herbal Treatment*, which contains an exhaustive list of conditions ranging from mild cases like the common cold to more serious cases such as malaria and stroke that can be cured with natural remedies. It states that "it

[1] https://play.google.com/store/apps/details?id=com.patrikat.stomachdiseasesandtreatment



62% 20:31

DermoApp

Hello User,

Diagnose your suspected lesion...

Diagnose now

Know about Melanoma Skin Cancer

9:37

277

Bitter Gourd

- Drink some bitter gourd juice on an empty stomach each morning.
- First remove the seeds of two to three bitter gourds and use a juicer to extract the juice.
- Add some water and then drink it.
- Follow this treatment daily in the morning for at least two months.

278

Cinnamon

- Mix one-half to one teaspoon of cinnamon in a cup of warm water. Drink it daily.
- Another option is to boil two to four cinnamon sticks in one cup of water and allow it to steep for 20 minutes. Drink this solution daily until you see improvement.

279

Figure 9.3 DermoApp (left); Home Remedies Herbal Treatment (right).

can help in curing 200+ of most common diseases" without providing any scientific substance or evidence to substantiate any claims.[2]

9.5 Suggested Improvements

The FDA's framework is designed to promote innovation in developing medical mobile apps, meaning less regulations and a focus on apps that can be physically problematic to patients [18]. While theoretically plausible, in practice, there are many applications that can be problematic. Additionally, a voluntary pathway to approval is unrealistic; most developers will not take the time, resources, and effort to seek FDA approval when the option to publish almost instantly on an app marketplace exists. The fact that HIPAA is only concerned with apps that work with covered entities is extremely narrow and leads to many apps easily bypassing any type of scrutiny. Apps which sell or distribute sensitive personal data to third parties without user consent should be penalized. The FTC intends to further improve and enforce their Breach Notification Rule as a priority and a means to ensure that entities that are not covered by HIPAA

[2] `https://play.google.com/store/apps/details?id=com.zhystudio.NaturalHomeRemedies&hl=en_US`

are held accountable [26]. However, mobile marketplaces should also be scrutinized and should promote stricter requirements for users developing medical apps. While the developer bears responsibility for ensuring that the apps is safe for public use and approved, they cannot be trusted to go through the lengthy process for approval. For instance, in the app descriptions found on Google Play for medical apps:

- 97 apps mention being HIPAA certified
- 14 apps stating that they are FDA approved
- 88 apps containing disclaimers that it is not a medical device or regulated

Additionally, Google Play itself does not have any strict guidelines regarding medical apps. The policies regarding health content and services are mainly limited to health misinformation, the sale and use of illegal substances and prescription drugs, and claimed app functionalities. Google states that apps which connect to external devices, such as an oximeter, must contain a disclaimer "that they are not intended for medical use, are only designed for general fitness and wellness purposes, are not a medical device, and must properly disclose the compatible hardware model/device model" [8].

Among thousands of medical apps available, we find it unlikely that these are the only apps requiring regulation or that are actually regulated. Regulatory entities should work with app marketplaces to ensure that apps, prior to publishing, are approved. One possibility is to integrate the FTC's set of questions for developers as part of the process of submitting a medical application.

9.6 Conclusions

As mobile health apps continue to expand and perform critical tasks, it is essential that users and their sensitive data are protected by local regulations. We detail regulatory entities based in the US to better understand which apps should be regulated. We automatically classify 192 apps that should be scrutinized by the FTC. We determine that the majority of apps most likely require some form of regulation and that current regulatory entities and app marketplaces need to be more diligent and stringent with approving medical apps for public usage.

CHAPTER 10

RELATED WORK

Related work regarding this dissertation falls into several categories: program analysis (Section 10.1), mobile malware detection (Section 10.2), suspicious behavior in mobile apps (Section 10.3), Android permissions (Section 10.4), medical mobile apps (Section 10.5), and mobile authentication (Section 10.6).

10.1 Static and Dynamic Analysis

As dynamic analysis and static analysis tools have been used in this dissertation, we discuss such tools that have been created and implemented by other researchers. Dynamic analysis tools analyze code as a program is being run, basing the analysis on runtime information. Dynamic analysis, however, only finds defects in code that has been executed. Additionally dynamic analysis tools are limited due to computational resources required to achieve an accurate analysis. Many researchers have attempted to improve dynamic analysis in the following ways. Andlantis is a scalable dynamic analysis tool able to process over 3000 Android apps per hour [45], reducing the overhead of dynamic analysis. Intellidroid configures inputs to a specific dynamic analysis tool [155], allowing to trigger malicious behavior that may be undetected by standard dynamic analysis. Using process traces or `ptrace`, DroidTrace monitors selected system calls of targeted processes, which run dynamic payloads, and classifies behaviors through system call sequences [160]. As a result, DroidTrace is able to detect different dynamic loading behaviors found in apps, especially malware.

TaintDroid is a dynamic taint analysis tool which monitors how apps access and manipulate personal data in realtime. Taintdroid was used with 30 randomly chosen popular Android apps and discovered 68 cases of potential data misuse in two-thirds of the apps studied [63].

Static analysis tools analyze an app's source code (bytecode) without running the app. Static analysis is exhaustive and approximates most aspects of app code that may not be present during runtime.

FlowDroid is a precise static taint analysis tool for Android apps that analyzes callbacks from the Android framework. By modeling call back methods and application lifecycles, FlowDroid is able to efficiently detect leaks without having a high false positive rate that is normally associated with static analysis [39].

Amandroid provides an alternative form static analysis for Android apps by tracking Android components and inter-component communication. As a result, Amandroid can address security issues that can result from multiple components' interactions within the same app or different apps [150].

Normally in taint analysis, data flow is tracked from sources to sinks and is presented as a source/sink pair. As a result, taint trackers are unable to handle complex concatenated sensitive identifiers that are usually processed in Android. Rahaman et al. introduce "algebraic" taint analysis that produces rich and expressive leak *signatures*, employing AND/XOR operators and exposing hashing operations [116]. Algebraic taint analysis is able to identify precisely which IDs are leaked, in what form, and how they are combined prior to leaking; their empirical study has revealed the leakiest libraries as well as the leakiest apps in a corpus of 1,000 top Google Play apps.

10.2 Mobile Malware Detection

As a means to document and characterize Android malware, Zhou et al. observed 1260 Android malware samples in 49 different families. The results indicated that roughly 86% of malware samples repackage legitimate apps, 37% contain platform level exploits to have more privileges, and 93% have bot-like capabilities. Additionally, they investigated the success of off-the-shelf commercial antivirus tools: 79.6% of malware were detected in best case scenarios while the worst case the tool could only detect 20.2% of malware [161].

Malware found in both official and unofficial Android markets have proven problematic. In a different study, Zhou et al. [162] present a too, DroidRanger, to efficiently detect potential infection from known and unknown malware found in five different Android marketplaces, including Google Play. DroidRanger uncovered 211 malicious apps and two zero day malware in all the Android app markets they studied.

As malware has spread throughout mobile systems, the security community becomes more inclined to employ automatic dynamic analyses. Consequently, malware authors have not stopped trying to detect these systems and to evade analysis. Vidas et al. [146] address various techniques that can detect whether an app is being analyzed, or running in a virtualized environment. They discuss four major categories of signals – analysis and virtualization indicators: behavior, performance, hardware and software components, and system design choices.

10.3 Detecting Suspicious Behavior on Mobile Apps

Monitoring the behavior of individual apps on Android is effective at exposing malware.

SpanDex is a set of extensions for Android's Dalvik virtual machine which ensures that apps do not leak user's passwords. Spandex handles implicit flows by using various techniques that quantify the amount of information a process' control flow reveals about a secret. SpanDex runs untrusted code in a data flow sensitive sandbox. Experiments on 50 popular Android apps discovered that, for 90% of users, an attacker is expected to need 80 or more login attempts to successfully guess their password [59].

ProfileDroid captures what apps actually do, while comparing with claimed app behavior, the resources utilized, entities the app communicates with, and executions of the same application [152].

AppsPlayground is a framework which attempts to automatically review and analyze smartphone apps. By integrating various components for detection, AppsPlayground evaluates the system using both benign and malicious apps. In addition, the implementation on roughly 4,000 apps resulted in a 25% improvement of testing. As a result, AppsPlayground is a scalable and automatic approach to tackle the growing number of apps in the app market [118].

10.4 Permissions in Android

Wei et al. studied the evolution of permissions in the Android ecosystem from Android 1.5 to Android 4.0.3 [151]. After discussing the different types of permissions required from various apps, Wei et al. mention the dangers of not only third party apps, but also of preinstalled apps – apps coming from phone manufacturers or mobile carriers. The evidence suggests that more and more apps wanted more dangerous permissions. Through these findings, the paper states that Android needed to improve the way permissions were handled to protect user privacy and security in a more proactive and user-friendly manner.

The tool Stowaway delves deeper in the permissions systems of Android 2.2 and detects over-privileged apps. Stowaway generates the maximum set of permissions an application requires and compares to the permissions that are actually being used. The paper concludes that the causes of over-privilege in many apps are developer error and confusion [65].

Apps using custom permissions rather than system permissions is another serious matter. Apps can have their functionality exposed or exploited when defining app-specific permissions. Overall the security implications were relatively unknown until Li et al [95] evaluated the design of custom permissions. Using the tool, CuPerFuzzer+, Li et al. were able to detect custom permission related vulnerabilities found in the system by treating the permission mechanism as a black box and executing massive targeted tests to trigger privilege escalation. As a result, 5,932 effective cases with 47 critical paths were discovered along with shortcomings in the Android permission framework.

Apps are also able to circumvent the permission system by using side channels and covert channels as discovered by Reardon et al. [119]. Apps can bypass permissions for READ PHONE STATE (which provides the IMEI of a device) and ACCESS NETWORK STATE (which provides the MAC address of an individual device). Furthermore, geolocation was acquired by apps via incoming ad mediation service packets which provided location data within the ad link using IP based geolocation. When apps circumvent permissions, the behavior is not inherently malicious or even intentional; however this successful circumvention reveals weaknesses within the Android system.

10.5 Medical Apps and Devices

Safety concern regarding medical mobile apps has been a prominent subject of study. Magrabi et al. [101] provide a commentary on the difficulties of regulating healthcare apps due to the fact anyone can make an app. Magrabi et al. confirm that there is little to no monitoring of use or formal evaluation of such apps. Mobile mental health apps are aimed to help patients manage their mental health conveniently on their mobile devices. However, there is an issue with the lack of clear regulations in mobile mental health apps as addressed by Terry et al. [135]. This paper creates a typology of the types of mobile mental health apps that exist. They also discuss the difficulty in judging the quality and efficacy of mental health apps, especially since many apps are developed outside of traditional healthcare spaces.

Developing safe medical apps and understanding the risks that apps entail have been a point of discussion and research. For example, Lewis et al. evaluate and create a risk framework of medical apps based on functionality [94]. Additionally, Wicks et al. provide methods on how one can develop a medical app safely and securely [153].

Several studies investigated the accuracy and overall usability of mobile health apps. Bierbrier et al.'s 2014 study [44] tested medical scores (including Child-Pugh and HAS-BLED) and calculations, e.g., BMI, in 14 apps (2 Android and 12 iOS) with 10 values: 2 extreme values and 8 middle values. There were errors in two Child-Pugh apps, though they were at the low score ranges hence would not place the patient over the threshold or in a different class. Coppetti et al. studied the accuracy of smartphone heart rate measurement apps and revealed that there were substantial performance differences between heart rate apps and clinical monitoring – as much as 20 beats per minute [58]. While the study focused on only 4 apps, app discrepencies may persist with other apps of this nature. The importance of responsible app marketplace safeguards regarding health apps is discussed by Wykes et al. [156]. Their work expressed concerns with the "overselling of health apps" and suggested a set of four principles that app marketplaces could use to guide the user to more sensible choices. Their study was conducted on four apps, rather than a larger dataset. Wisniewski et al.'s study on top-rated health apps confirmed our findings that most medical apps "continue to have no scientific evidence to support their use" [154]; their study is based on manual analysis of 120 apps. Tangari et al. [132] discovered severe privacy issues in 88% of medical apps used in their study, namely that medical apps could potentially share user data with third party advertising and tracking services. Other studies show there is little evidence to whether health apps work, finding that only a small fraction of apps is tested [52], leading to suggestions of "prescribed health apps", or having health apps vetted by medical professionals as a prescription rather than being able to be freely installed. The reliability and safety of health apps is discussed by Akbar et al. with most concerns stemming from the quality of content presented in apps, such as presenting incorrect and incomplete information [30]. Regarding weight loss apps, Zaidan et al. addressed the usability features of these apps by applying an evaluation framework [157]. The framework revealed that app marketplace search engines had biases towards certain titles and keywords that did not reflect the full functionality of the app and that the most popular apps are not necessarily the most effective. Brown et al.'s review of 76 pregnancy apps regarding nutrition determined that such apps should not be considered as an appropriate resource for pregnant women due to unsound nutritional advice and overall unreliability [50].

10.6 Mobile Authentication

The security of various mobile authentication systems has been widely investigated and studied. Research has shown that users tend towards predictable and popular choices, regardless of the authentication method. Bonneau et al. [46] studied 4-digit PINs and concluded that while 4-digit PINs fare better in user choices, guessing a birthday is an effective strategy to access a user's account. Wang et al. showed that 6-digit PINs have marginally better security than 4-digit PINs, yet both English and Chinese users fall into certain patterns when choosing PINs [149]. Markert et al. collected PINs specifically primed for mobile authentication and demonstrated that 6-digit PINs offer little (and perhaps worse) benefit over 4-digit PINs against a throttled attacker. Moreover, non-enforcing blocklists (as deployed by iOS) do not increase security [103]. Patterns, or graphical passwords, have been studied in multiple contexts, including smudge attacks [42], shoulder-surfing [68, 62, 102, 41], and user strength perceptions [35, 34]. The selection process has also been studied [143, 40, 128], and in all cases, users choices are predictable. Zhang et al. attempt to remedy these vulnerabilities by changing the shape of patterns. Instead of a 3 x 3 layout, schemes that have stronger security than the original Android pattern are considered [158]. Along with security, usability is an important facet of authentication method adoption, thus, quantifying user feedback of such methods is pertinent [124]. Regarding biometric adoption and perceptions, users considered biometrics to be more secure than PINs according to Bhagavatula et al. [43]. In addition, usability factors (such as poor lighting for facial recognition) contributed to users' negative feedback and reluctance to adopt this method versus a more convenient method such as fingerprint recognition. Even biometrics can lead to users choosing weaker forms of knowledge-based authenticators [54]. Zhang et al. offer a solution by combining biometrics with voice and face recognition [159].

CHAPTER 11

CONCLUSIONS AND FUTURE WORK

11.1 Conclusions

To conclude, we see how mobile devices are an integral part to modern society, and how important it is to ensure that mobile apps are safe and secure. We provided a discussion of the Android ecosystem, including apps. The broad scope of mobile security, as we detail, extends from device authentication to deceptive app behavior. We explained how device administrator systems can be exploited and characterize medical apps. We showed how medical apps have a wide range of functions and cater to wide audiences, but also may overstate their actual functionalities in order to attract users. We looked into whether medical score calculators are accurate or not and discovered that both implementations and specifications of critical scores have errors. Finally, we discussed regulatory frameworks and enforcement for medical apps.

11.2 Future Work

In this dissertation, we have seen the broad scope of mobile security and how apps can exploit user data without their knowledge. Additionally we have seen the broad landscape of apps, especially in the Medical category, and the variety of functionalities they may have and how they can affect users. With these concepts in mind, there are further studies that can be completed and addressed. Supplementing our observations on medical app regulations in the US, we intend on expanding our study to better refine and find apps which lack federal approval and need regulation. We hope to create a fully automatic process that can be implemented by Google Play and other app marketplaces that can force developers of medical apps to go through the full approval process. Last, we intend on expanding our regulatory framework by studying other countries' and institutions' regulations and searching for possible loopholes and improvements that can be made.

BIBLIOGRAPHY

[1] Activity: Android developers. https://developer.android.com/reference/android/app/Activity#ActivityLifecycle as of December 22, 2022.

[2] Android 12 is running on 13.3% of all devices ahead of android 13 launch. https://9to5google.com/2022/08/12/android-12-distribution-numbers/.

[3] App manifest overview. https://developer.android.com/guide/topics/manifest/manifest-intro?hl=enas of December 22, 2022.

[4] App resources overview. https://developer.android.com/guide/topics/resources/providing-resources?hl=enas of December 22, 2022.

[5] Apple reinvents the phone with iphone. https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/as of December 22, 2022.

[6] Application fundamentals. https://developer.android.com/guide/components/fundamentals?hl=enas of December 22, 2022.

[7] F-droid - free and open source android app repository. https://f-droid.org/.

[8] Health content and services - play console help. https://support.google.com/googleplay/android-developer/answer/12261419?hl=en#zippy=\%2Cexamples-of-common-violations.

[9] In the matter of docket no. flo health, inc. https://www.ftc.gov/system/files/documents/cases/flo_health_complaint.pdf.

[10] International classification of diseases (icd). https://www.who.int/standards/classifications/classification-of-diseases.

[11] Mobile app download statistics usage statistics (2021). https://buildfire.com/app-statistics/as of December 22, 2022.

[12] Permissions and apis that access sensitive information - play console help. https://support.google.com/googleplay/android-developer/answer/9888170?hl=en&ref_topic=9877467 as of December 22, 2022.

[13] Platform architecture : Android developers. https://developer.android.com/guide/platform.

[14] Remembering the first android phone, the t-mobile g1 (htc dream). https://www.androidauthority.com/first-android-phone-t-mobile-g1-htc-dream-906362/as of December 22, 2022.

[15] Samsung galaxy store. https://galaxystore.samsung.com/apps.

[16] Services overview: Android developers. https://developer.android.com/guide/components/services#Lifecycle.

[17] tfidf :: A single-page tutorial - information retrieval and text mining. http://www.tfidf.com/.

[18] The regulatory perspective q&a with fda's bakul patel, 2012. https://www.aami.org/docs/default-source/uploadedfiles/filedownloads/horizons/regulatory-perspective.pdf.

[19] What is android fragmentation, and can google fix it?, Sep 2016. https://www.androidauthority.com/android-fragmentation-google-fix-it-713210/ as of December 22, 2022.

[20] Announcing the launch of the android partner vulnerability initiative, Oct 2020. https://security.googleblog.com/2020/10/announcing-launch-of-android-partner.html.

[21] Ftc cracks down on marketers of "melanoma detection" apps, Sep 2021. https://www.ftc.gov/news-events/news/press-releases/2015/02/ftc-cracks-down-marketers-melanoma-detection-apps.

[22] Mobile os market share 2021, Jun 2021. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/#statisticContainer as of December 22, 2022.

[23] National early warning score (news) 2, Nov 2021.

[24] Biggest app stores in the world 2022, Nov 2022. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[25] MEWS Brasil, March 2022. https://play.google.com/store/apps/details?id=appinventor.ai_blinkeado.InformaticasaudeMEWS.

[26] Mobile health apps and the ftc's health breach notification rule: New enforcement initiative coming, Jan 2022. https://www.natlawreview.com/article/mobile-health-apps-and-ftc-s-health-breach-notification-rule-new-enforcement.

[27] Policy for device software functions and mobile medical applications guidance for industry and food and drug administration staff, Sep 2022.

[28] Doug Aamoth. First smartphone: Fun facts about simon, Aug 2014. https://time.com/3137005/first-smartphone-ibm-simon/as of December 22, 2022.

[29] Saba Akbar, Enrico Coiera, and Farah Magrabi. Safety concerns with consumer-facing mobile health applications and their consequences: A scoping review. *Journal of the American Medical Informatics Association*, 27(2):330–340, 2019.

[30] Saba Akbar, Enrico Coiera, and Farah Magrabi. Safety concerns with consumer-facing mobile health applications and their consequences: a scoping review. *J. Am. Med. Inform. Assoc.*, 27(2):330–340, February 2020.

[31] Chloe Albanesius. Google 'bouncer' now scanning android market for malware, Feb 2012. `https://uk.pcmag.com/mobile-apps/66697/google-bouncer-now-scanning-android-market-for-malware`.

[32] Mamdouh Alenezi and Iman Almomani. Abusing android permissions: A security perspective. *2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, 2017.

[33] Wayne M. Alves, Brett E. Skolnick, Brett E. Skolnick, and Shamik Chakraborty. *Chapter 5 - Traumatic Brain Injury.* Academic Press, 2018.

[34] Panagiotis Andriotis, Theo Tryfonas, and George Oikonomou. Complexity Metrics and User Strength Perceptions of the Pattern-Lock Graphical Authentication Method. In *Conference on Human Aspects of Information Security, Privacy and Trust*, HAS '14, pages 115–126. Springer, Heraklion, Crete, Greece, June 2014.

[35] Panagiotis Andriotis, Theo Tryfonas, George Oikonomou, and Can Yildiz. A Pilot Study on the Security of Pattern Screen-Lock Methods and Soft Side Channel Attacks. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 1–6, Budapest, Hungary, April 2013. ACM.

[36] Android Open Source Project. Device admin deprecation, August 2018. `https://developers.google.com/android/work/device-admin-deprecation`.

[37] Android Open Source Project. Android enterprise, July 2019. `https://developers.google.com/android/work/overview`.

[38] Android Open Source Project. Android notifications overview: Status bar and notification drawer, July 2019. `https://developer.android.com/guide/topics/ui/notifiers/notifications\#bar-and-drawer`.

[39] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 259–269, New York, NY, USA, 2014. ACM.

[40] Adam J. Aviv, Devon Budzitowski, and Ravi Kuber. Is Bigger Better? Comparing User-Generated Passwords on 3x3 vs. 4x4 Grid Sizes for Android's Pattern Unlock. In *Annual Computer Security Applications Conference*, ACSAC '15, pages 301–310, Los Angeles, California, USA, December 2015. ACM.

[41] Adam J. Aviv, John T. Davin, Flynn Wolf, and Ravi Kuber. Towards Baselines for Shoulder Surfing on Mobile Authentication. In *Annual Conference on Computer Security Applications*, ACSAC '17, pages 486–498, Orlando, Florida, USA, December 2017. ACM.

[42] Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith. Smudge Attacks on Smartphone Touch Screens. In *USENIX Workshop on Offensive Technologies*, WOOT '10, pages 1–7, Washington, District of Columbia, USA, August 2010. USENIX.

[43] Chandrasekhar Bhagavatula, Blase Ur, Kevin Iacovino, Su Mon Kywey, Lorrie Faith Cranor, and Marios Savvides. Biometric Authentication on iPhone and Android: Usability, Perceptions, and Influences on Adoption. In *Workshop on Usable Security*, USEC '15, San Diego, California, USA, February 2015. ISOC.

[44] Rachel Bierbrier, Vivian Lo, and Robert C Wu. Evaluation of the accuracy of smartphone medical calculation apps. *J. Med. Internet Res.*, 16(2):e32, February 2014.

[45] Michael Bierma, Eric Gustafson, Jeremy Erickson, David Fritz, and Yung Ryn Choe. Andlantis: Large-scale android dynamic analysis. *arXiv preprint arXiv:1410.7751*, 2014.

[46] Joseph Bonneau, Sören Preibusch, and Ross Anderson. A Birthday Present Every Eleven Wallets? The Security of Customer-Chosen Banking PINs. In *Financial Cryptography and Data Security*, FC '12, pages 25–40, Kralendijk, Bonaire, February 2012. Springer.

[47] Michael J. Breslow and Omar Badawi. Severity scoring in the critically ill. *Chest*, 141(1):245–252, 2012.

[48] John Brooke. SUS: A Quick and Dirty Usability Scale. *Usability Evaluation in Industry*, pages 189–194, 1996.

[49] Amy Brown, Apoorva Ballal, and Mo Al-Haddad. Recognition of the critically ill patient and escalation of therapy. *Anaesthesia Intensive Care Medicine*, 20(1):1–5, 2019.

[50] Hannah M. Brown, Tamara Bucher, Clare E. Collins, and Megan E. Rollo. A review of pregnancy apps freely available in the google play store. *Health Promotion Journal of Australia*, 31(3):340–342, 2019.

[51] Published by Statista Research Department and Jul 6. Healthcare apps available google play 2021, Jul 2021. `https://www.statista.com/statistics/779919/health-apps-available-google-play-worldwide/`.

[52] Oyungerel Byambasuren, Sharon Sanders, Elaine Beller, and Paul Glasziou. Prescribable mhealth apps identified from an overview of systematic reviews. *npj Digital Medicine*, 1(1), 2018.

[53] Check Point Software Technologies. Check point capsule connect, December 2018. https://play.google.com/store/apps/details?id=com.checkpoint.CloudConnector.

[54] Ivan Cherapau, Ildar Muslukhov, Nalin Asanka, and Konstantin Beznosov. On the Impact of Touch ID on iPhone Passcodes. In *Symposium on Usable Privacy and Security*, SOUPS '15, pages 257–276, Ottawa, Canada, July 2015. USENIX.

[55] Catalin Cimpanu. Play store identified as main distribution vector for most android malware, Nov 2020.

[56] Comscore, Inc. Top OEMs - Share of Smartphone Subscribers 3 Month Avg. Ending Nov. 2019 vs. 3 Month Avg. Ending Sep. 2019, September 2019. https://www.comscore.com/Insights/Rankings#tab_mobile_smartphone_oems, as of December 22, 2022.

[57] Contributor. Guest post: Symbian os – one of the most successful failures in tech history, Nov 2010. https://techcrunch.com/2010/11/08/guest-post-symbian-os-one-of-the-most-successful-failures-in-tech-history-2/guccounter=1&guce_referrer=aHR0cDovL3d3dy5kaWdpdGFsbWFya2V0aW5nbWF0dXJpdHltb2RlbC5jb20v&guce_referrer_sig=AQAAAFdDYK_F0WoqQxYYmXTND0ehMSdARE0Do1_48QxBQfdcco1JY1nRwB7piLaBlZqZ9GA7vvdr0oNMi6cj94Xrf-Z5W2LqCXmPbXvAldtH8HdJ4q70P9rz-kLZMlm_EqgVSpEpAWvVcqefZTq-507lMNp02rljZuXLW_vyKjUF4P3t.

[58] Thomas Coppetti, Andreas Brauchlin, Simon Müggler, Adrian Attinger-Toller, Christian Templin, Felix Schönrath, Jens Hellermann, Thomas F Lüscher, Patric Biaggi, Christophe A Wyss, and et al. Accuracy of smartphone apps for heart rate measurement. *European Journal of Preventive Cardiology*, 24(12):1287–1293, 2017.

[59] Landon P Cox, Peter Gilbert, Geoffrey Lawler, Valentin Pistol, Ali Razeen, Bi Wu, and Sai Cheemalapati. Spandex: Secure password tracking for android. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 481–494, 2014.

[60] Simon Crossley. Eu regulation of health information technology, software and mobile apps. https://uk.practicallaw.thomsonreuters.com/2-619-5533?contextData=(sc.Default).

[61] Denis Crăciunescu. A short history of mobile malware, Nov 2020. https://proandroiddev.com/a-short-history-of-mobile-malware-296570ed5c1b as of December 22, 2022.

[62] Alexander De Luca, Marian Harbach, Emanuel von Zezschwitz, Max-Emanuel Maurer, Bernhard Ewald Slawik, Heinrich Hussmann, and Matthew Smith. Now You See Me, Now You Don't: Protecting Smartphone Authentication from Shoulder Surfers. In *ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2937–2946, Toronto, Ontario, Canada, April 2014. ACM.

[63] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick Mcdaniel, and Anmol N. Sheth. Taintdroid. *ACM Transactions on Computer Systems*, 32(2):1–29, 2014.

[64] FDA. Examples of software functions for which the fda will exercise enforcement discretion, December 2019. `https://www.fda.gov/medical-devices/device-software-functions-including-mobile-medical-applications/examples-software-functions-which-fda-will-exercise-enforcement-discretion`.

[65] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638, 2011.

[66] Flavio Lopes Ferreira. Serial evaluation of the sofa score to predict outcome in critically ill patients. *JAMA*, 286(14):1754, 2001.

[67] Greir Ander Huck Flynn, Barbara Polivka, and Jodi Herron Behr. Smartphone use by nurses in acute care settings. *Comput. Inform. Nurs.*, 36(3):120–126, March 2018.

[68] Alain Forget, Sonia Chiasson, and Robert Biddle. Shoulder-Surfing Resistance with Eye-Gaze Entry inCued-Recall Graphical Passwords. In *ACM Conference on Human Factors in Computing Systems*, CHI '10, pages 1107–1110, Atlanta, Georgia, USA, April 2010. ACM.

[69] FTC. Mobile health apps interactive tool, December 2019. `https://www.ftc.gov/tips-advice/business-center/guidance/mobile-health-apps-interactive-tool`.

[70] J Gardner-Thorpe, N Love, J Wrightson, S Walsh, and N Keeling. The value of modified early warning score (mews) in surgical in-patients: A prospective observational study. *The Annals of The Royal College of Surgeons of England*, 88(6):571–575, 2006.

[71] Annette M Green. Kappa statistics for multiple raters using categorical classifications. In *Proceedings of the 22nd annual SAS User Group International conference*, volume 2, page 4, 1997.

[72] Ames Gross. Column - medical software regulations in asia 2017, Nov 2017. `https://www.medtechintelligence.com/column/medical-software-regulations-asia-2017` as of December 22, 2022.

[73] John Gruzelier. Unwanted effects of hypnosis: A review of the evidence and its implications. *Contemporary Hypnosis*, 17(4):163–193, 2000.

[74] Hana Habib, Jessica Colnago, William Melicher, Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Password Creation in the Presence of Blocklists. In *Workshop on Usable Security*, USEC '17, San Diego, California, USA, February 2017. ISOC.

[75] Faye Haffey, Richard R. Brady, and Simon Maxwell. A comparison of the reliability of smartphone apps for opioid conversion. *Drug Safety*, 36(2):111–117, 2013.

[76] Marian Harbach, Emanuel von Zezschwitz, Andreas Fichtner, Alexander De Luca, and Matthew Smith. It's a Hard Lock Life: A Field Study of Smartphone (Un)Locking Behavior and Risk Perception. In *Symposium on Usable Privacy and Security*, SOUPS '14, pages 213–230, Menlo Park, California, USA, July 2014. USENIX.

[77] HHS. Breach notification rule, December 2019. `https://www.hhs.gov/hipaa/for-professionals/breach-notification/index.html`.

[78] HHS. Health information laws, December 2019. `https://www.hhs.gov/hipaa/for-professionals/security/index.html`as of December 2019.

[79] HHS. Summary of privacy rule, December 2019. `https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html`as of December 2019.

[80] Jerry Hildenbrand. What's a kernel?, Jan 2012. `https://www.androidcentral.com/android-z-what-kernel` as of December 22, 2022.

[81] Eveline Hitti, Dima Hadid, Jad Melki, Rima Kaddoura, and Mohamad Alameddine. Mobile device use among emergency department healthcare professionals: Prevalence, utilization and attitudes. *Scientific Reports*, 11(1), 2021.

[82] Kit Huckvale, Samanta Adomaviciute, José Tomás Prieto, Melvin Khee-Shing Leow, and Josip Car. Smartphone apps for calculating insulin dose: a systematic assessment. *BMC Med.*, 13(1):106, May 2015.

[83] Kit Huckvale, Samanta Adomaviciute, José Tomás Prieto, Melvin Khee-Shing Leow, and Josip Car. Smartphone apps for calculating insulin dose: A systematic assessment - bmc medicine, May 2015. `https://bmcmedicine.biomedcentral.com/articles/10.1186/s12916-015-0314-7`.

[84] Sybase Inc. Sap mobile secure for android, March 2019. `https://play.google.com/store/apps/details?id=com.Android.Afaria` as of December 22, 2022.

[85] Christian Johnson and Wesley Barker. Hospital capabilities to enable patient electronic access to health information, 2019, Jun 2021. `https://s3.documentcloud.org/documents/20796639/hospital_capabilities_to_enable_patient_access_onc-data-brief.pdf` as of December 22, 2022.

[86] Michael Kan. Android malware abuses app permissions to hijack phones, Dec 2019. https://www.pcmag.com/news/android-malware-abuses-app-permissions-to-hijack-phones as of December 22, 2022.

[87] Anastasiya Kharychkova. How to publish android app on google play store in 10 steps, May 2021. https://orangesoft.co/blog/how-to-publish-an-android-app-on-google-play-store.

[88] Rahul Khosla. New data shows 57% of shoppers prefer mobile apps to other channels. https://www.heady.io/blog/new-data-shows-57-of-shoppers-prefer-mobile-apps-to-other-channels as of December 22, 2022.

[89] W A Knaus, E A Draper, D P Wagner, and J E Zimmerman. APACHE II: a severity of disease classification system. *Crit. Care Med.*, 13(10):818–829, October 1985.

[90] Knock Lock. Knock Lock Screen - Applock, 2020. https://play.google.com/store/apps/details?id=com.knocklock.applock&hl=en_US, as of December 22, 2022.

[91] Ivana Križanović. Cell phone history: From the first phone to today's smartphone wonders, Nov 2021. https://versus.com/en/news/cell-phone-history as of December 22, 2022.

[92] Federica Laricchia. Global mobile os market share 2022, Nov 2022. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[93] Mark Erik Larsen, Kit Huckvale, Jennifer Nicholas, John Torous, Louise Birrell, Emily Li, and Bill Reda. Using science to sell apps: Evaluation of mental health app store quality claims. *npj Digital Medicine*, 2(1), 2019.

[94] Thomas Lorchan Lewis and Jeremy C Wyatt. mhealth and mobile medical apps: A framework to assess risk and promote safer use. *J Med Internet Res*, Sep 2014.

[95] Rui Li, Wenrui Diao, Zhou Li, Shishuai Yang, Shuang Li, and Shanqing Guo. Android custom permissions demystified: A comprehensive security evaluation. *IEEE Transactions on Software Engineering*, pages 1–1, 2021.

[96] Gregory Y. Lip and Deirdre A. Lane. Stroke prevention in atrial fibrillation. *JAMA*, 313(19):1950, 2015.

[97] Gregory Y.H. Lip, Lars Frison, Jonathan L. Halperin, and Deirdre A. Lane. Comparative validation of a novel risk score for predicting bleeding risk in anticoagulated patients with atrial fibrillation: The has-bled (hypertension, abnormal renal/liver function, stroke, bleeding history or predisposition, labile inr, elderly, drugs/alcohol concomitantly) score. *Journal of the American College of Cardiology*, 57(2):173–180, 2011.

[98] Marte Løge, Markus Dürmuth, and Lillian Røstad. On User Choice for Android Unlock Patterns. In *European Workshop on Usable Security*, EuroUSEC '16, Darmstadt, Germany, July 2016. ISOC.

[99] Adele Lubell. Potentially dangerous exercises: Are they harmful to all? *The Physician and Sportsmedicine*, 17(1):187–192, 1989.

[100] MaaS360. Maas360 mdm, December 2018. `https://play.google.com/store/apps/details?id=com.fiberlink.maas360.android.control`.

[101] Farah Magrabi, Ibrahim Habli, Mark Sujan, David Wong, Harold Thimbleby, Maureen Baker, and Enrico Coiera. Why is it so difficult to govern mobile apps in healthcare? *BMJ Health & Care Informatics*, 26(1), 2019.

[102] Shushuang Man, Dawei Hong, and Manton Matthews. A Shoulder-Surfing Resistant Graphical Password Scheme – WIW. In *International Conference on Security and Management*, SAM '03, pages 105–111, Las Vegas, Nevada, USA, June 2003. CSREA Press.

[103] Philipp Markert, Daniel V. Bailey, Maximilian Golla, Markus Dürmuth, and Adam J. Aviv. This PIN Can Be Easily Guessed: Analyzing the Security of Smartphone Unlock PINs. In *IEEE Symposium on Security and Privacy*, SP '20, pages 1525–1542, San Francisco, California, USA, May 2020. IEEE.

[104] William Melicher, Darya Kurilova, Sean M. Segreti, Pranshu Kalvani, Richard Shay, Blase Ur, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Michelle L. Mazurek. Usability and Security of Text Passwords on Mobile Devices. In *ACM Conference on Human Factors in Computing Systems*, CHI '16, pages 527–539, Santa Clara, California, USA, May 2016. ACM.

[105] Patrick Mutchler, Yeganeh Safaei, Adam Doupe, and John Mitchell. Target fragmentation in android apps. *2016 IEEE Security and Privacy Workshops (SPW)*, 2016.

[106] Naveeninfotech. Mobile tracker, December 2018. `https://play.google.com/store/apps/details?id=com.nav.mobile.tracker`.

[107] Dor Nizar. F5 labs investigates malibot, Jun 2022. `https://www.f5.com/labs/articles/threat-intelligence/f5-labs-investigates-malibot`.

[108] Royal College of Physicians. National early warning score (news) 2: standardising the assessment of acute-illness severity in the nhs. updated report of a working party 2017. 2021.

[109] T. Olsson, A. Terent, and L. Lind. Rapid emergency medicine score: A new prognostic tool for in-hospital mortality in nonsurgical emergency department patients. *Journal of Internal Medicine*, 255(5):579–587, 2004.

[110] Ortholive. 32 Statistics on mHealth, April 2019. https://www.ortholive.com/blog/32-statistics-on-mhealth/.

[111] Charlie Osborne. This is how malicious android apps avoid google's security vetting, Oct 2019. https://www.zdnet.com/article/this-is-how-malicious-android-apps-avoid-googles-security-vetting/ as of December 22, 2022.

[112] Rikesh K Patel, Adele E Sayers, Nina L Patrick, Kaylie Hughes, Jonathan Armitage, and Iain Andrew Hunter. A UK perspective on smartphone use amongst doctors within the surgical profession. *Ann. Med. Surg. (Lond.)*, 4(2):107–112, June 2015.

[113] Timothy B. Plante, Anna C. O'Kelly, Bruno Urrea, Zane T. MacFarlane, Roger S. Blumenthal, Jeanne Charleston, Edgar R. Miller, Lawrence J. Appel, and Seth S. Martin. User experience of instant blood pressure: Exploring reasons for the popularity of an inaccurate mobile health app. *npj Digital Medicine*, 1(1), 2018.

[114] Emil Protalinski. Android malware numbers explode to 25,000 in june 2012, Jul 2012. https://www.zdnet.com/article/android-malware-numbers-explode-to-25000-in-june-2012/ as of December 22, 2022.

[115] Emil Protalinski. Google play removed 700,000 bad apps in 2017, 70% more than in 2016, January 2018. https://venturebeat.com/2018/01/30/google-play-removed-700000-bad-apps-in-2017-70-more-than-in-2016/.

[116] Sydur Rahaman, Iulian Neamtiu, and Xin Yin. Algebraic-datatype taint tracking, with applications to understanding android identifier leaks. *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.

[117] Android Intelligence By JR Raphael and JR Raphael. Exclusive: Inside android 4.2's powerful new security system, Nov 2012.

[118] Vaibhav Rastogi, Yan Chen, and William Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220, 2013.

[119] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 603–620, 2019.

[120] Claire Reilly. Goodbye, windows 10 mobile, tweets joe belfiore, Oct 2017. https://www.cnet.com/tech/mobile/windows-10-mobile-features-hardware-death-sentence-microsoft/.

[121] K J Rhee, C J Fisher, Jr, and N H Willitis. The rapid acute physiology score. *Am. J. Emerg. Med.*, 5(4):278–282, July 1987.

[122] Shanon Roberts. Top 10 most downloaded apps of 2021 so far. https://www.cyberclick.net/numericalblogen/top-10-most-downloaded-apps-of-2020-so-far.

[123] 2009 5:32 am UTC Ryan Paul Feb 24. Dream(sheep): A developer's introduction to google android, Feb 2009. https://arstechnica.com/gadgets/2009/02/an-introduction-to-google-android-for-developers/.

[124] Florian Schaub, Ruben Deyhle, and Michael Weber. Password Entry Usability and Shoulder Surfing Susceptibility on Different Smartphone Platforms. In *International Conference on Mobile and Ubiquitous Multimedia*, MUM '12, pages 13:1–13:10, Ulm, Germany, December 2012. ACM.

[125] Hannah L Semigran, Jeffrey A Linder, Courtney Gidengil, and Ateev Mehrotra. Evaluation of symptom checkers for self diagnosis and triage: audit study. *BMJ*, 351, 2015.

[126] A. J. Six, B. E. Backus, and J. C. Kelder. Chest pain in the emergency room: Value of the heart score. *Netherlands Heart Journal*, 16(6):191–196, 2008.

[127] Sharon R. Smith, Jack D. Baty, and Dee Hodge. Validation of the pulmonary score: An asthma severity score for children. *Academic Emergency Medicine*, 9(2):99–104, 2002.

[128] Youngbae Song, Geumhwan Cho, Seongyeol Oh, Hyoungshick Kim, and Jun Ho Huh. On the Effectiveness of Pattern Lock Strength Meters: Measuring the Strength of Real World Pattern Locks. In *ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2343–2352, Seoul, Republic of Korea, April 2015. ACM.

[129] Damjan Jugović Spajić, Bojan Jovanović, and Ivana Vojinovic. Mobile banking statistics that show wallets are a thing of the past, Jun 2021. https://dataprot.net/statistics/mobile-banking-statistics/.

[130] Tom Spring. Google's war on android app permissions, 60 percent successful. https://threatpost.com/googles-war-on-android-app-permissions-60-percent-successful/153311/.

[131] 9to5 Staff. Jobs' original vision for the iphone: No third-party native apps, Oct 2011. https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/.

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

[133] Team Counterpoint. US Smartphone Market Share: By Quarter, November 2019. https://www.counterpointresearch.com/us-market-smartphone-share/, as of December 22, 2022.

[134] G Teasdale and B Jennett. Assessment of coma and impaired consciousness. a practical scale. *Lancet*, 2(7872):81–84, July 1974.

[135] Nicolas P. Terry and Tracy D. Gunter. Regulating mobile mental health apps. *Behavioral Sciences the Law*, 36(2):136–144, 2018.

[136] Tesseract-Ocr. tesseract-ocr/tesseract: Tesseract open source ocr engine (main repository). `https://github.com/tesseract-ocr/tesseract`.

[137] LG Thailand. Lg knock code™ vs fingerprint scanner, 2015. `https://www.youtube.com/watch?v=0Imk5JILUc0`.

[138] LG Thailand. Lg knock code™ vs pin code, 2015. `https://www.youtube.com/watch?v=NRInfu-Lhnc`.

[139] Daniel R. Thomas, Alastair R. Beresford, Thomas Coudray, Tom Sutcliffe, and Adrian Taylor. The lifetime of android api vulnerabilities: Case study on the javascript-to-java interface. *Security Protocols XXIII Lecture Notes in Computer Science*, page 126–138, 2015.

[140] Kevin C. Tofel. LG G2 and G Flex Phones Getting the Knock Code Wake and Unlock Feature, March 2014. `https://gigaom.com/2014/03/25/lg-g2-and-g-flex-phones-getting-the-knock-code-wake-and-unlock-feature/`, as of December 22, 2022.

[141] Tom Watkins. Why it's time for enterprises to adopt android's modern device management apis, December 2017. `https://www.blog.google/products/android-enterprise/why-its-time-enterprises-adopt-androids-modern-device-management-apis/`.

[142] Cody Toombs. Meet art, part 1: The new super-fast android runtime google has been working on in secret for over 2 years debuts in kitkat, Nov 2013. `https://www.androidpolice.com/2013/11/06/meet-art-part-1-the-new-super-fast-android-runtime-google-has-been-working-on-in-secret-for-over-2-years-debuts-in-kitkat/`as of December 22, 2022.

[143] Sebastian Uellenbeck, Markus Dürmuth, Christopher Wolf, and Thorsten Holz. Quantifying the Security of Graphical Passwords: The Case of Android Unlock Patterns. In *ACM Conference on Computer and Communications Security*, CCS '13, pages 161–172, Berlin, Germany, November 2013. ACM.

[144] U.S. Census Bureau, Population Division. Annual Estimates of the Resident Population by Single Year of Age and Sex for the United States: April 1, 2010 to July 1, 2018 , 2018 Population Estimates, June 2019. `https://factfinder.census.gov/bkmk/table/1.0/en/PEP/2018/PEPSYASEXN?#`, as of December 22, 2022.

[145] U.S. Census Bureau, Population Division. Annual Estimates of the Resident Population for Selected Age Groups by Sex for the United States, States, Counties, and Puerto Rico Commonwealth and Municipios: April 1, 2010 to July 1, 2018, June 2019. `https://factfinder.census.gov/bkmk/table/1.0/en/PEP/2018/PEPAGESEX?#`, as of December 22, 2022.

[146] Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 447–458, 2014.

[147] Jean-Louis Vincent, Rui Moreno, Jukka Takala, S Willatts, A Mendonça, H Bruining, C Reinhart, Peter Suter, and L Thijs. The sofa (sepsis-related organ failure assessment) score to describe organ dysfunction/failure. on behalf of the working group on sepsis-related problems of the european society of intensive care medicine. *Intensive care medicine*, 22:707–10, 08 1996.

[148] Sean Wallace, Marcia Clark, and Jonathan White. 'it's on my iphone': attitudes to the use of mobile computing devices in medical education, a mixed-methods study. *BMJ Open*, 2(4):e001099, August 2012.

[149] Ding Wang, Qianchen Gu, Xinyi Huang, and Ping Wang. Understanding Human-Chosen PINs: Characteristics, Distribution and Security. In *ACM Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 372–385, Abu Dhabi, United Arab Emirates, April 2017. ACM.

[150] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. Amandroid. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[151] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 31–40, 2012.

[152] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Profiledroid: Multi-layer profiling of android applications. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 137–148, 2012.

[153] Paul Wicks and Emil Chiauzzi. 'trust but verify' – five approaches to ensure safe medical apps. *BMC Medicine*, 13(1), 2015.

[154] Hannah Wisniewski, Gang Liu, Philip Henson, Aditya Vaidyam, Narissa Karima Hajratalli, Jukka-Pekka Onnela, and John Torous. Understanding the quality, effectiveness and attributes of top-rated smartphone health apps. *Evidence Based Mental Health*, 22(1):4–9, 2019.

[155] Michelle Y Wong and David Lie. Intellidroid: A targeted input generator for the dynamic analysis of android malware. In *NDSS*, volume 16, pages 21–24, 2016.

[156] Til Wykes and Stephen Schueller. Why reviewing apps is not enough: Transparency for trust (t4t) principles of responsible health app marketplaces. *Journal of Medical Internet Research*, 21(5), 2019.

[157] Sarah Zaidan and Erin Roehrer. Popular mobile phone apps for diet and weight loss: A content analysis. *JMIR mHealth and uHealth*, 4(3), 2016.

[158] Lei Zhang, Yajun Guo, Xiaowei Guo, and Xiaowei Shao. Does the layout of the android unlock pattern affect the security and usability of the password? *Journal of Information Security and Applications*, 62:103011, 2021.

[159] Xinman Zhang, Dongxu Cheng, Pukun Jia, Yixuan Dai, and Xuebin Xu. An efficient android-based multimodal biometric authentication system with face and voice. *IEEE Access*, 8:102757–102772, 2020.

[160] Min Zheng, Mingshen Sun, and John C.S. Lui. Droidtrace: A ptrace based android dynamic analysis system with forward execution capability. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 128–133, 2014.

[161] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.

[162] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In *NDSS*, volume 25, pages 50–52, 2012.

[163] Heinz Zimmermann and Jürg Reichen. Hepatectomy: Preoperative analysis of hepatic function and postoperative liver failure. *Digestive Surgery*, 15(1):1–11, 1998.