Auditing and Attributing Behaviours of Suspicious Android Health Applications

Muhammad Salman
Macquarie University
Sydney, Australia
muhammad.salman@mq.edu.au

I Wayan Budi Sentana
Macquarie University
Sydney, Australia
i-wayan-budi.sentana@students.mq.edu.au

Muhammad Ikram*
Macquarie University
Sydney, Australia
muhammad.ikram@mq.edu.au

Mohamed Ali Kaafar
Macquarie University
Sydney, Australia
dali.kaafar@mq.edu.au

ABSTRACT

Mobile health and fitness applications for consumers, collectively known as mobile health applications or mHealth, monitor user activities such as steps, locations, and email. It seamlessly aggregates sensitive information to facilitate a wide range of functions, such as the management of health conditions and symptom checking. Although mHealth apps provide real-time health monitoring and easier access to healthcare resources, they can also pose serious risks to user safety. Although the research community is primarily well aware of the user's exposure to several types of malware, there has not been a large-scale in-depth analysis of suspicious mHealth apps using a consistent methodology.

This study conducts a large-scale security and privacy analysis of 381 suspicious free mHealth apps (chosen from a corpus of 15,893 apps) available on "Google Play". We built a customized toolset to perform a comprehensive analysis of these applications. We explore the range of mechanisms used by mHealth apps to monitor users' activities, such as photos, text messages, and live microphone access, mainly through the injection of suspicious third-party libraries. In addition, we uncover the use of obfuscation methods used by suspicious mHealth apps to hide their malicious codes. As mHealth apps are used by a large number of customers worldwide, we argue that patients, clinicians, technology developers, and policy-makers alike should be conscious of the hidden risks involved and weigh them carefully against the benefits.

1 INTRODUCTION

With the steady growth in the population's access to smartphone devices, we have witnessed an explosion of mobile applications (in short, apps) available through various online marketplaces. As of late 2020, approximately 2.56 million apps [45] are available on Google Play alone. Breaking these by category, we note two popular, mutually exclusive categories of Medical and Health & Fitness apps. Referred to as mobile health (or mHealth) apps, these encompass a range of functions, from chronic condition management and symptom checkers to step/calorie counters and period trackers [48]. Reflecting the growth of this app segment, recent guidelines from the U.S. Food and Drug Administration (FDA) formalised the use of mHealth apps for healthcare and recommended considering those providing aid to patients or clinicians as medical devices [34].

*Corresponding author.

As a result of recent advances in integrating new technologies, mobile phones are equipped with many useful sensors, which enable mHealth apps to offer a wide range of features and functionalities, such as recording users' health-related information and seamlessly monitoring users' behaviours and activities. Examples of such activities are weight, smoking or drinking trackers. Another class of such apps is used to contact your doctor, book appointments, vaccinations, medication and get e-prescriptions [73]. Moreover, health-related information collected and saved by these applications is of sensitive nature, and the security and privacy of health-related data are of particular importance [70].

Mobile health applications (mHealth apps) by design track, manage and store the health data of users [83]. Mobile devices and apps used in health-related practises allow different forms of security and privacy threats [71] and dangers such as decentralised data storage systems, unwanted access to all the data at times and not enough regulations etc. So, several security and privacy challenges and relevant requirements need to be addressed [64]. In this study, we conducted the first large-scale analysis of suspicious mHealth apps available on Google Play with the goal of attributing the malicious behaviour, especially the behaviour related to user information harvesting and sharing, to the third party.

We developed custom-built test suites to analyse the source code and app's behaviour during the runtime of 381 suspicious mHealth apps on Google Play. Our analysis highlights a range of security and privacy audits from the characterisation of the malware family, attribution of malicious behaviour in mHealth apps that leads us to the finding of malicious third-party library and Online App Generator involvement from the perspective of the mHealth app's user. Our main findings are listed below:

- 75 (19.68) % of suspicious mHealth apps embed modified or malicious third-party libraries, including 14 apps detected to obfuscate their libraries.
- 114 (29.92%) suspicious mHealth developer conceal the malicious code inside of the apps built on the Online App Generator framework.
- 335 (88%) suspicious mHealth invoking methods to harvest user's information to be sent to third-party servers.
- 50 (13.12%) suspicious mHealth embedding library illegally harvested users' information through legitimate libraries such as Google and Facebook.

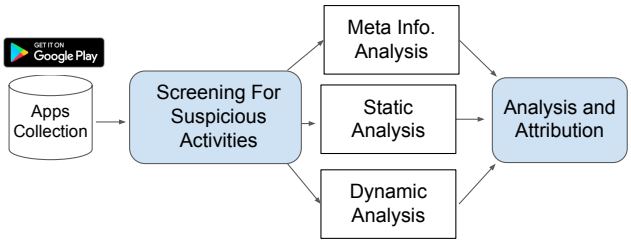

Figure 1: Overview of our methodology for collecting and analysis suspicious mHealth apps on Google Play. Our analysis pipeline consists of three different types of analysis (meta info, static, and dynamic) to audit suspicious activities in the analysed mHealth apps.

- 88 (23.09%) suspicious mHealth currently available and detected malicious by VirusTotal.

We believe this work further sharpens the understanding of suspicious mHealth apps, providing guidance for phone operating vendors and regulators in their efforts to undermine the use and accessibility of such applications.

2 OVERVIEW OF MOBILE HEALTH APPLICATION

2.1 Types of mHealth Apps

Portability, advanced hardware and sophisticated applications have made a cell phone highly suitable for mHealth applications. The landscape of health-related apps is vast and comes with various advantages and disadvantages. In this section, we look at the state-of-the-art health applications available on Google Play.

Apps for doctors and physicians. This is a class of highly sophisticated applications that are not necessarily free. Nurses and health practitioners use these apps to take advantage of drug referencing tools, clinical decision support tools, centralized health record system access, access to medical information materials, and updating and sharing of patient records. One of the most popular such apps is drug-reference guides such as 'Epocrates', 'Lexicomp', 'Wolters Kluwer' and clinical-decision support reference tool apps such as 'UpToDate', and 'Medscape' [20].

Disease specific applications. These mobile apps are designed for single diseases, such as "Eye Handbook", an app for children's health safety "Outbreaks near me" and apps in orthopaedics etc [66] [46].

Applications for the general public. The most common types of mHealth apps are patient-centered apps. These applications are capable of performing comparatively trivial but a wide range of functions, helping consumers or users with managing exercise or yoga routines, keeping a record of how much a person walks, sleeps, or drinks water, managing chronic disease, weight, and smoking [63] [76].

3 DATA AND ANALYSIS METHODOLOGY

Collecting mHealth Apps from Google Play. Google Play neither provides a complete list of mHealth apps nor does its search functionality yield all the available apps. To overcome this and detect as many mHealth apps as possible, we developed a crawler that interacted directly with the store's interface. Starting from the top-100 apps from the Medical and Health & Fitness categories on Google Play, the crawler systematically searched through other apps considered similar by Google Play (i.e., other apps presented on Google Play apps' pages in the section 'Similar apps' that belong to the same category). For each app, the crawler collected the following metadata: app category and price, locations where the app is available, app description, number of installs, developer information, user reviews, and app rating. Overall, we discovered 15,893 unique mHealth apps on the Google Play store as of January 1, 2021. Next, we use Raccoon [21] to download the APKs of the corresponding apps, which are then scanned with VirusTotal(VT) [90], as explained further in the following.

Scanning mHealth apps with VirusTotal. To identify suspicious mHealth apps, we inspected the APKs using the VirusTotal public API, which aggregates the scanning capabilities of 68 popular antivirus tools. For each analyzed app, we obtained the malware label and several malware positives showing the class of malware, such as trojan or adware, and the agreement among the antivirus tools in classifying the app as malware. VirusTotal has been commonly used to detect malicious apps, executables, software and domains [42–44, 49]. As each of VirusTotal's antivirus tools may produce false positives, we computed the aggregate AV-Rank metric, i.e., the number of tools that flagged an APK as malicious, with the maximal score being 68. To minimize the occurrence of false positives and obtain a clear indication of malicious activity, we restricted our further analyses to apps having an AV-Rank ≥ 2, in agreement with previous studies on Android app malware [14, 44].

For each analysed mHealth app VT generates a report that highlights whether any given AV tool has detected an app as malicious with a detection label. We aggregate labels for each mHealth app and leverage the [77][53] and Euphony [41][33] to identify the types and families of malware present in its source code.

The results from VirusTotal revealed a number of apps which are exceptionally malicious in nature. Of the 381 apps analysed, 130 (34.2%) have an *AV-rank* ≥ 2 The detection labels from VirusTotal showed that the apps consist of five main types of malware. Adware was detected in 45% (172) of the apps, followed by Trojan in 25% (97). Riskware, Malware, and Grayware combined were found in a total of 9% (34) of the apps. The remainder of the apps were either in an undefined category or returned a vague and undeterminable label from VirusTotal. Table 1 lists the Top 10 *suspicious* mHealth apps based on their respective *AV-rank*. We then use the scanning results to analyze the capabilities (explained below) and behavior of the 381 mHealth apps.

Static Analysis. We analyze the source code for each suspicious mHealth app using our custom-built tools set. In particular, we report on the app' requesting sensitive permission analysis, the presence of tracking libraries in the app's source code decompiled using `Apktool` [6]. Apktool decompiles Dalvik bytecode into Java bytecode and converts `.dex` and `classes.dex` files to `.smali`using smali disassembler. We access `AndroidManifest.xml` and parse `uses-permission` and `service` tags to extract app permissions for resources and data on a user's smartphone.

Table 1: List of *Suspicious* mHealth apps with *AV-rank* ≥ 18.

App Title	AV-rank	Installs	Rating
Urdu Best Totkays	27	10,000+	3.8
EEMCQ	22	100+	N/A
Nursing Care Plans List	20	5,000+	4.6
Nursing Diagnosis and Care Plans FREE	20	10,000+	4.4
Aapak Totkay	20	100,000+	3.8
Mental Health and Psychiatric Nursing Care Plans	19	1,000+	4.1
Fit Bites	19	10,000+	4.1
YOGA FOR DIABETES	18	10,000+	3.4
TritionRx: Tube Feeding	18	1,000+	3.6

We analyze the extracted permissions for potential illegitimate requests. mHealth apps often need sensitive permissions to work. Permissions like `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION` may be questionable in non-mHealth apps, but they may be appropriate in mHealth apps to improve step counting or pedometer accuracy. The permissions of the Top 20 benign mHealth apps from Google Play's Top 10 *Health & Fitness* and Top 10 *Medical* apps are analysed to determine if the requested permissions are due to malicious behaviour or mHealth functionality. Different mHealth apps may require various dangerous permissions. For comparative analysis, suspicious mHealth apps are divided into *Health & Fitness* and *Medical* categories. As a way to flag apps with potentially dangerous behaviours, the permissions sought by *suspicious* mHealth apps are compared to the permissions typically asked by malicious apps [93][30][54].

To determine if apps' requested permissions are necessary, we map each app's permission to Android To this end, we leverage method-to-permission mappings techniques proposed by PScout [19] and Johnson et al. [47]. In particular, we examine the `smali` files to determine which calls launch permission-protected Android API calls. Any permissions not in the API calls were deemed excessive. This reveal which suspicious mHealth apps request the permissions that they require and which requests are excessive permissions. Once the API calls are mapped to the apps' permission, we then perform manual checks to determine their origin. This is disclosed if the permissions are asked by the app itself or a third-party library embedded in the source code. Lastly, a dictionary of malicious signatures (including malware domain URLs) obtained during malware detection is established together with known malware signatures [15], to help detect the origin of suspicious app behaviour. This is used to search for malware signature calls or invoke related methods.

`Apktool` expands third-party libraries in the apps whereas Python's `os.walk` function extracted the libraries and subdirectories. Apps that have several `classes.dex` files are expanded into distinct directory trees, which are then traversed to extract the third-party libraries. After collecting the libraries, they are compared to a dictionary of tracking libraries compiled from prior research [44]. The functioning of any libraries not listed in the dictionary was determined by manually analysing and researching them. Libraries that were discovered to be malicious or unknown were flagged as *suspicious*, and their functionality was further investigated.

Dynamic Analysis. To perform dynamic analysis on the apps, we execute them in an observable environment. Android Studio supports virtual Android phones for this purpose. Most of the trials used a virtual Google Pixel running Android OS 8.0. However, due to varying app requirements, some were run on older Android phones and versions of Android OS. For behaviours to be triggered in the apps, their activities and components must be interacted with. To automate this procedure and increase the number of interactions, the input generator MonkeyRunner [60] was utilized. Moreover, `mitmproxy` (an interactive HTTPS proxy) [58] was used to monitor and record the content of each request, as depicted in Figure 2. In particular, we analyze the content of POST requests as they are used to exfiltrate data to external servers. We examine POST requests to ascertain the exact data communicated between a given analyzed app and its external servers. We also examine whether the analyzed applications encrypt communications via services like HTTPS to protect against in-path attacks.

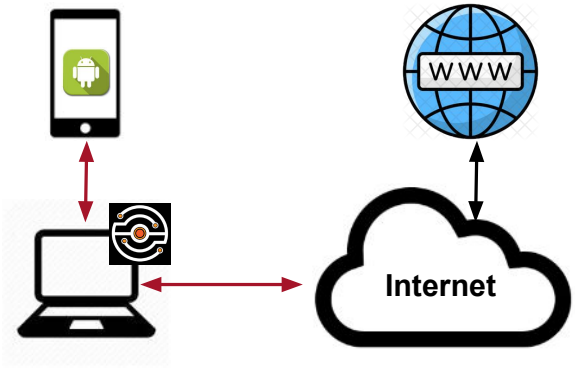

Figure 2: Overview of our testbed for the analysis of the runtime, and network behaviors of (suspicious) apps.

This investigation focused on whether apps communicated with malicious domains and what information was shared with third-party domains. The network packets were decoded in order to obtain the requested URLs and domains. Using VT's URL scanner, the security of each domain was evaluated to identify if apps transmitted or received data from malicious domains.

Metadata Analysis We leverage metadata that we collected from the Google Play store to characterize the behaviors of suspicious mHeath apps. In particular, for each suspicious mHealth app, our crawler collects metadata, including name, description, version, category, size, latest update date, rating, number of ratings and installs, minimum Android version, and developer details. Using the app's description, we leverage CoreNLP (a natural language processing tool) [56] to determine the main declared functionality of the app on Google Play. Moreover, we use app's average rating and the number of installs to characterize the popularity of apps on Google Play. We also use user reviews text to characterize users' perceptions of suspicious mHealth apps.

4 CHARACTERIZING MALWARE FAMILIES

In this section, we characterise the malicious code presence in the analysed mHealth apps. First, we identify and classify the malware family of each suspicious mHealth app. Since the VirusTotal returns information based on each provider's standard, we then aggregate the malware families and types based on the machine learning

aggregator for the malware family conducted by [77][53] and Euphony [41][33]. To increase the confidence of malicious behaviour, we only considered apps with an AV-rank ≥ 5 in our analysis. The results for the malware family and type classification are provided in the Table 2.

Table 2: Malware families detected in *Suspicious* mHealth apps, ordered by number of apps infected.

Family	Type	# of Apps	App Example	# of Install	AV Rank
Airpush	Adware	77	Uses for Coconut Oil	100,000+	16
Leadbolt	Adware	8	Aapak Totkay	100,000+	20
Revmob	Adware	5	Sleep Analyzer	100,000+	13
Autoins	Trojan	4	Simvalley PhoneWatch	100,000+	8
AppsGeyser	Adware	2	Clickpharmacy	-	13
SmsReg	Trojan	2	Madarsho	50,000+	10
Viser	Adware	1	Urdu Best Totkays	10,000+	27
Buzztouch	Trojan	1	EEMCQ	100+	22

Table 2 shows that the malware type is dominated by adware and Trojan. Antivirus tools in the VirusTotal report label with the string "a variant of" for a particular family name, indicating that the malware is modifying an original or a legit code or library. Our other result found that several malware family names returned by VirusTotal replicate the original or legit library and have the following characteristics.

Airpush: "A variant of Airpush" malware is leveraging the Airpush notification library that is used to embed the advertisement library into the apps. Originally, Airpush is a legit library used to monetize apps. However, a malware developer modifies this library to display an excessive ad in their app. During the runtime, this malware loads `libjiagu-1004670200` dynamic library and harvests the user's location, device type, MAC address, and network operator information. Our dynamic analysis also found that this malware sends the Android ID that is used as a device identifier during ads monetization. Other than the dynamic library, this malware tainted a temporary database transaction in `webview.db-journal` indicating that the apps open an SQLite database in `Webview`. As a note, Android creates a file with the "-journal" sufix for every temporary SQLite transaction conducted on a device. To protect its binary, this malware obfuscates its library and leverages a customized name, although analysis of their method headers and class structures reveals that they are obfuscated versions of the legit advertising library.

Leadbolt: Similar to Airpush, LeadBolt also invokes several geographical or location tagging methods such as `getCountry()`, `getLongitude()` and `getLatitude()` enabled by `ACCESS_FINE_LOCATION` permission. LeadBolt also requested the privilege to access the devices' logs and execute shell scripts to harvest CPU maximum and minimum frequency information. A tainted temporary SQLite transaction is also found as `ua.db-journal` file crafted under the app's directory. As confirmed by [92], this malware can take over the current display in the devices. Based on our observation and results returned by VirusTotal, the name of this malware is similar to the Leadbolt monetizing library provided by [50]. Hence, we suspect this malware impersonates or modifies the original library.

Appsgeyser: Originally, AppsGeyser is an Online App Generator (OAG) that allows non-programming end-user to create their own mobile apps [65]. However, in this study, we observe that the suspicious mHealth app developer has modified the original AOG platform to conduct malicious activity. The variant of Appsgeyser malware is detected by Microsoft, ESET, and Quick-Heal anti-virus engines. In contrast, other engines such as NANO and Avira detect this variant as a Potentially Unwanted Application (PUA) or FakeApps variant. In this study, we found this malware injected into two apps. Analysis of the network traffic found that both apps communicated with five different malicious domains and made a POST request to the malicious domain, such as "adaranth.com" in which it shared the device's screen dimensions, country code, latitude, and longitude of the user's device.

Revmob: It is a Brazilian-based mobile ad network company that provides a service library for mobile app monetization. Unfortunately, this library is no longer supported due to the company's operational issues. We then traced the online code repository and found several GitHub repositories related to the Revmob ad monetization library. We suspect that the five suspicious mHealth apps that were detected contained the Revmob malware, had duplicated and modified the library, and used it in their applications. Fortiguard labelled this malicious library as "adware" because it displays advertising content to the user. Typically, the display appears as pop-up advertisements and takes over the current Android display without user consent [36].

Autoins: This malware was found in 4 suspicious mHealth apps and detected by eight antivirus engines. However, we could not find more detailed information about this malware from each antivirus repository. The eight antiviruses only mention that Autoins is a Trojan-type malware. Interestingly, we found several reports in [25, 89] revealing that the Autoins variant was found in pre-installed apps on Android devices manufactured in East Asia and Europe. This trojan variant is an auto-updater known as `Android/PUP.Riskware.Autoins.Redstone`.

SMSreg: Fortiguard classifies this malware as Trojan [35], while Microsoft flagged it as a Potentially Unwanted Application (PUA) [57] and some other engines marked it as Grayware or Riskware [31]. Currently, there is still a debate about whether this kind of library can be categorized as malware or not. Regarding SMSreg, several discussions on the Malwarebyte forum show that this library is used to perform auto registration via SMS. One of the apps uses it to do billing via SMS. Some apps that adopt this library have been whitelisted by Malwarebytes [55].

Viser: The app injected by Viser family malware request six dangerous permissions, including `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION`, and `READ_PHONE_STATE`. The obfuscated folder name is found and contains *Vserv Mobi* sub-directory. The API call analysis shows that *Vserv Mobi* invokes numerous methods, including SMS-related activities and obtaining the device's exact coordinates, Device Type, ID, and MAC Address. Additional calls were found to invoke the `DownloadManager` of the `android.app` API to dynamically fetch additional app components. The network traffic analysis found multiple requests to the "vserv.mobi" domain and shared the user's Time Zone, Latitude, Longitude, Screen Dimensions, Advertising ID, Serial Number, and IMEI.

Buzztouch: Like AppsGeyser, Buzztouch is an Online App Generator(OAG). Other than the OAG service, Buztouch also provides various plugin services for iOS and Android. However, in this study, we believed that the suspicious mHealth detected to contain Buzztouch malware was crafted by Buztouch OAG. Other than the *Facebook* third-party library, this apps only consist of the main directory called *v1_4* with sub-directory *eemcq*, which matches the App's ID. This indicates that the app did not embed any Buzztouch plugin library. Analysis of the App's API calls found that the *v1_4* directory invokes many calls that are used to harvest information related to the device coordinates, Last Known Location, Device ID, Phone Number, and Serial Number. Further analysis of the API calls also found that the library uses the `java.net` API to invoke the `openConnection()` method. This call's parameter is an HTTP request to the "www.buzztouch.com" domain. The query of this GET request includes the device's ID, brand, model, and latitude and longitude.

5 SUSPICIOUS BEHAVIOUR ATTRIBUTION

In this section, we leverage our analysis (cf. § 3) of the source code of each suspicious mHealth app to audit and attribute any suspicious behaviour. In particular, we report on the potential misuse of apps' capabilities exhibited by requested permissions and the integration of third-party libraries. We also present our analysis of suspicious mHealth apps piggybacking online app generators to generate malware at low development costs rapidly. We also elaborate on apps' abilities to harvest and exfiltrate users' sensitive health-related data to external servers and the mechanisms employed to hide their potentially suspicious behaviours. Finally, we present byte entropy analysis to validate the injection and integration of malware in the source.

5.1 Misusing Permissions for Potential Exploits

Due to the significant number of dangerous permissions requested by both the *suspicious* and benign mHealth apps, determining an app as *malicious* solely based on the dangerous permissions it requests was not feasible. As such, the permissions of *suspicious* apps were compared with permissions that are commonly found in malware. This also revealed which permissions have the potential for misuse. The apps can execute malicious code silently or take over the entire phone screen with permissions granted.

Table 3: Overview of permissions potentially misused by *suspicious* mHealth apps.

Type	Name	# of Apps	Possible misuse
Dangerous	`WRITE_EXTERNAL_STORAGE`	244	Store malicious data
	`READ_PHONE_STATE`	129	Tracking user devices
	`CALL_PHONE`	40	Make phone call without permission
Normal	`INTERNET`	370	Download malicious scripts
	`WAKE_LOCK`	239	Run malicious code continuously
	`VIBRATE`	120	Disable vibrator and notifications
	`RECEIVE_BOOT_COMPLETED`	116	Execute malicious code when booted
	`GET_TASKS`	48	Monitor and discover private info
Signature	`SYSTEM_ALERT_WINDOW`	31	Display ads over other apps
	`READ_LOGS`	20	Access sensitive data

5.2 Abusing Third-Party Libraries Privilege

Guided by the results of the malware characterization in section 4, we then took a deeper look at the mHealth apps directory. During our observations, we found similarities between the name of the malware family and several popular commercial advertisement libraries used to monetize mobile apps, such as Revmob and Airpush. However, we found inconsistencies between the use of this advertisement library and VirusTotal's findings. For example, we found that 27 mHealth apps embed Revmob libraries in their apps (see Table 4). However, VirusTotal's detection results show that only five apps (see Table 2) are infected with this type of malware. Based on this, we believe the five applications detected containing viruses resulted from modifications from the legit Revmob library. Our further research found that the Revmob library is publicly available in the GitHub repository,[1] hence modifications to this library can be conducted efficiently.

Another indication that enhances our suspicion that mHealth apps modify legit libraries is shown in the adoption of the Airpush library. The 77 apps containing the Airpush malware family have varying AV-ranks, with 16 being detected by more than ten anti-virus engines. We then observed more detail and found that the 16 apps were obfuscated in certain directories by renaming the Smali file into shorter names such as `a.smali, a$1.smali, b.smali, etc`. Each obfuscated library has a unique name; however, analysis of their method headers and class structures reveal that they are obfuscated versions of the same advertising library. Further analysis reveals numerous calls which invoke methods related to Airpush, including a method called `getAirPushAppId()`, and calls that construct URLs for the "ads.airpush.com" domain. The network traffic of these apps found that frequent GET requests were made to the "apportal.airpush.com" domain. Additionally, ten apps shared critical device and user data with the "api.airpush.com" and "www.pushnotificationsender.com" domains. These include the device's brand, model, IMEI, time zone, locale, country, and exact latitude and longitude.

Originally, *Airpush* (rebrand into *Airnow*) is a legitimate advertising platform that assists Nissan, Walmart, KFC, and Huawei with advertising [3]. We believe that the suspicious mHealth developers have abused this library to gain the privilege of push notifications owned by this library, to excessively loading advertisement content to increase the monetization revenue of the apps.

Except for `Qihoo` library, Table 4 shows all modified third-party libraries categorised as advertisement or analytics supporting advertisement libraries. `Qihoo` is an Internet security company that provides a wide range of protection, including Anti-virus, binary-protector and various security plug-ins for web and mobile-based applications [24, 85]. Our API calls analysis revealed that the library dynamically loads an additional library called `Jiagu` which invokes the `chmod` native command on a file in the `Jiagu` library. This command is used to change the access permissions of file system objects to launch the `Jiagu` Packer platform. This finding is consistent with the result returned by APKiD in the sub-section 5.6, where two mHealth apps are embedding Packer to protect their binary files.

[1] https://gist.github.com/revmob-sdk/3383267

Table 4: 3rd-party libraries cause suspicious behaviours (left) and Online App Generator (right) in *Suspicious* mHealth apps, ordered by number of apps injected.

Modified Third-party Library			Online App Generator (OAG)		
Name	**#of Apps(%)**	**AV Rank**	**Name**	**#of Apps(%)**	**AV Rank**
Revmob	27 (7.07%)	1 to 10	Seattleclouds	65 (17.06%)	1 to 16
Umeng	16 (4.19%)	1 to 4	Appinventor	25 (6.56%)	1 to 16
Airpush	16 (4.19%)	1 to 10	AppsVision	9 (2.36%)	11 to 17
Leadbolt	8 (2.09%)	1 to 10	Mobincube	9 (2.36%)	1
WQMobile	2 (0.52%)	14	AppsGeyser	6 (1.57%)	1 to 10
Qihoo	2 (0.52%)	16	Buzztouch	1 (0.26%)	22
Vserv Mobi	1 (0.26%)	27			
Rever	1 (0.26%)	9			
Gmobi	1 (0.26%)	8			
Total	**75 (19.68%)**		**Total**	**114 (29.92%)**	

5.3 Piggybacking Online App Generators (OAGs)

During static analysis, we observe a particular naming pattern on several mHealth apps. Our further research find that the applications are created using Online App Generators (OAGs) or Online App Builders. The OAGs provide an online platform for the end-user with low-code to non-code skills to create their mobile apps [65]. An OAG provides various advantages, including support for multiple platforms so that the apps can run on iOS or Android, support for different monetizing components, and even the publishing pipeline.

To observe the existence of OAG used to create suspicious mHealth apps, we then conduct OAG fingerprinting through a specific indicator. Several platforms, such as `Appsvision` and `Mobincube` can be easily identified from the Package ID due to the package naming convention. While the others, such as `Seattlecloud` and `Appsgeyser` required more effort of decompilation and running through the app's directories. The result of OAG fingerprinting in Table 4 shows that Seattlecloud is dominating the OAG usage to build suspicious mHealth apps with 65 apps, followed by Appinventor and Appsgeyser with 25 and 9 apps, respectively.

Research in [65] discovered that the OAG platforms are vulnerable to specific attacks, including reconfiguration and infrastructure attacks. In this study, we find the real case of how OAG platforms failed to protect their products so they could be utilized to conduct malicious activities. Table 4 shows that at least 114 suspicious mHealth apps leverage the OAG platforms to conceal their malicious code. We believe this malicious behaviour is not inherited from the OAG platform because each app has non-uniform behaviour. As a sample, we then traced the existence of `Appsvision` and find that at least 485 mHealth apps were built on this platform. However, from this number, we only determine nine apps detected to contain malware by VirusTotal.

Moreover, the AV-rank of apps developed by the same platform also shows various numbers, indicating the non-uniformity of mHealth malicious behaviour. Another case of customized malicious behaviour of OAG-based apps is also shown by suspicious mHealth apps built on AppsGeyser, where only two apps (Table 2) were detected to contain Appsgeyser malware by VirusTotal out of 6 suspicious mHealth apps (Table 4) developed on the Appsgeyser OAG platform. In addition, we found that only 2 out of 65 apps leveraging Seattlecloud downgraded their connection protocol to plain HTTP, indicating that the flaw was not inherited from the original OAG platform.

Since the Android OS requires all applications running under its environment are packaged in the form of an APK file, piggybacking malware code into the OAG platforms is even more reasonable. OAG allows malware developers to package applications without developing their apps from scratch, considering that malicious code is generally only a small part of the entire application package.

5.4 Information Harvesting and Sharing

Based on our methods to permission mapping (cf. § 5.1), we observe that suspicious mHealth apps massively harvested user and device information. As shown in Table 5, 88% (335) of mHealth apps invoke `getPhoneType()`, `getDeviceId()`, `getNetworkOperatorName()` and `getNetworkCountryIso()` methods under the `TelephonyManager` class to request such information through `android.telephony` API. We then trace the request's source and found that most of the methods are invoked by third-party libraries, such as Seattleclouds, Pollfish, Tencent, Startapp, Tappx, and Truene, categorized as Targeted Advertising Library by [44].

The harvested information is then sent to the associated third-party library's server. During the dynamic analysis, we found at least 32% (121) of the apps share the information with advertising domains through the use of POST requests. Analysis of the requested content revealed that the app shared information about the device, such as the device type, the screen dimensions, and the country code the phone is set to. Additionally, the exact latitude and longitude of the device were found in the URL query. Although this measurement result is considered to be lower bound due to technical issues such as SSL pinning adoption, this data harvesting and sharing are worrisome.

Data harvesting and information sharing are typically conducted by the advertising library to create a user profile and used to set up a corresponding ad that matches the user profile. Having a match-targeted user can increase the possibility of ad hits to monetize the apps. The monetisation from the ads hit can be a source of funding for the app's developer since all the suspicious mHealth apps are free to download.

To check whether the information harvested is also shared with malicious domains, we tested all the domains accessed by mHealth apps against VirusTotal. As a result, we found that 21 apps shared the information with 22 domains flagged as malicious and phishing sites, including `adaranth.com`, `ds-club.ru`, `live.chartboost.com` and `cdn2.editmysite.com`. We also capture the request-response from `ds-club.ru` returned JavaScript which loads an Ad and sets its visibility to *hidden*.

In addition, we found that 36 apps shared data in plain text through unencrypted HTTP requests. The content in these POST requests ranged from sharing data about the screen dimensions, OS version, and device model to sensitive data such as the Device ID, Serial Number, MAC Address, and the country of the device.

5.5 Cross Library Data Harvesting

Due to massive method invocations related to user information harvesting, we were suspicious if the legit third-party library did

Table 5: Dominant API calls, methods and class invocation detected in the source codes of suspected mHealth apps.

API Call	Class	Method	# of Apps
android.media	MediaRecorder		107 (28%)
	AudioRecord		25(7%)
android.hardware	SensorManager		339 (89%)
	Camera		307 (81%)
android.telephony	TelephonyManager	getPhoneType()	335 (88%)
		getDeviceId()	335 (88%)
		getNetworkOperatorName()	335 (88%)
		getNetworkCountryIso()	335 (88%)
	SmsManager	sendTextMessage()	45(12%)
		sendMultipartTextMessage()	45(12%)
	PackageManager	getInstalledPackages()	72(19%)
		getInstalledApplications()	103(27%)
java.util	Locale	getCountry()	367 (96%)
		getLatitude()	334(88%)
		getLongitude()	334(88%)
	LocationManager	getLastKnownLocation()	268(70%)
android.accounts	AccountManager		199 (52%)

not just trigger the invocation. Hence, in this study, we also figure out the appearance of third-party libraries that illegally collect users' information from the legit libraries installed on devices such as Facebook and Google. This type of data gathering mechanism is called Cross Library Data Harvesting (XLDH) [91]. This library actively monitors the package manager to find a targeted legit library installed on the device. Those libraries then illegally harvest the user's data by capturing the information flow during the information interchange via API. Due to this illegal and malicious operation, Facebook has taken legal action against the XLDH library provider [91].

Since we were struggling to intercept information flown in HTTPS tunnel and breaking the certificate authority of the SSL pinning mechanism, we traced the appearance of the XLDH library in mHealth apps by relying on the previous research of [91]. Table 6 shows that 50 mHealth apps embed the XLDH library in their apps. The trace results of Revmob and Umeng XLDH libraries are aligned with the number of libraries found to be malicious in Table 4. Hence, we believe this XLDH library could be a source of malicious behaviour detected by VIRUSTOTAL.

5.6 Adopting Evasive and Obfuscation Methods

We determine whether an app uses any means of evading, obscuring, or disrupting the analysis of parties other than the application developers. In fact, malware developers rely on these techniques to evade primary analysis layers of application market stores such as Google Play [23]. On average 52% of its malware samples leverage anti-analysis techniques to evade, obscure, or disrupt analysis methods [40]. In Appendix A, we provide a detailed analysis of the six different types of evasion techniques employed by the analysed mHealth apps.

5.7 Byte Entropy Analysis

Elastic Malware Benchmark for Empowering Researcher (EMBER) [5] is popular among researchers as a benchmark dataset for Windows portable executable files. The EMBER dataset contains nearly 1.1 million samples. The LIEF project is severed as a backbone framework to extract features from PE files such as (*i*) General file information, (*ii*) Header information, (*iii*) Imported functions, (*iv*) Exported functions, (*v*) Section information, (*vi*) Byte histogram, (*vii*) Byte entropy histogram, and (*viii*) String information. The raw features are extracted and stored in JSON format, normalised into a feature vector for automated analysis such as classification.

Inspired by EMBER, we extend the static analysis to APKs as follows:

(1) Obtain VDEX and ODEX files by installing APK on Android devices. The VDEX and ODEX files were introduced by Google to optimise running time and boost time for APK apps in the Android environment.
(2) Utilise the LIEF framework to extract raw features from VDEX and ODEX files [1].
(3) Convert the raw features into vectors for automated analysis and classification.

To categorise the samples by Binary Protection Type, we can characterise them based on byte entropy (see Figure 3). Malware authors usually encrypt or compress their applications to conceal their malicious intent from scanners. These methods transform the application body into a series of random-looking data bytes. An efficient way to detect the encryption or packer trails in an application is to calculate the statistical variation of byte data in a fixed block length (e.g. 256 bytes.

Figure 3) depicts that the suspicious samples have the same pattern with different high and low entropy on specific byte positions. This information confirms the observation above that the suspicious applications were built using a few common online app builders (see § 5.3). We observe that 114 (29.9%) applications with "malicious"

Table 6: Malicious Cross Library Data Harvesting (XLDH) library detected in suspicious mHealth apps.

XLDH Library	Exfiltrated Data	Exfiltration Endpoints	# of Apps (%)
com.revmob	Facebook AccessToken	https://android.revmob.com	27(7.08%)
com.umeng.socialize	Facebook/Twitter/Dropbox/Kakao/Yixin/Wechat/ QQ/Sina/Alipay/Laiwang/Vk/Line/Linkedin's AccessToken and user data (ID/name/link/photo)	http://plbslog.umeng.com/umpx_share	6 (1.57%)
com.inmobi	Google activity	https://sdkm.w.inmobi.com/user/e.asm	4 (1.05%)
com.appfireworks	goole id, android id	http://api.appfireworks.com/t/	3 (0.78%)
com.yandex.metrica	Google Advertising id, Android id	https://startup.mobile.yandex.net	3 (0.78%)
cn.sharesdk	Bytedance	http://api.share.mob.com/log4	2 (0.52%)
com.ad4screen	Facebook appid, AccessToken	http://api.ad4s.local	2 (0.52%)
com.appsgeyser	Google Advertising id, Android id, IMEI, Mac Address	https://ads.aerserv.com/as/sdk/v3/	2 (0.52%)
com.oneaudience	Facebook id, name, gender, email, link, Twitter user data	https://api.oneaudience.com/api/devices	1 (0.26%)
		Total	50 (13.12%)

activities were added manually after the apps were generated. In addition, there are 14 applications with obfuscated activities, which explains why the obfuscation bars have higher entropy than others.

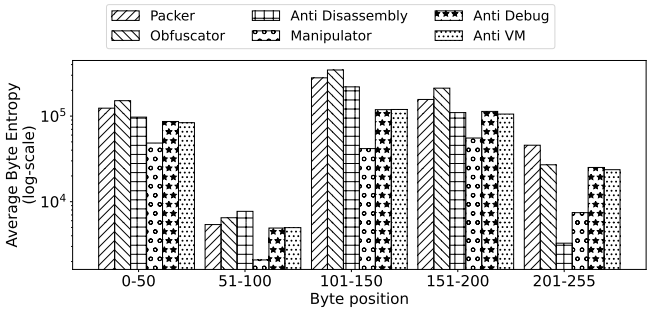

Figure 3: Byte entropy comparison of evasive and obfuscation methods (cf. Appendix A) employed by suspicious mHealth apps.

6 STATUS QUO AND USER AWARENESS

Status Quo. Recently, Google conducted a major restructuring of its Play store, requiring developers to upload and update their apps in Android App Bundle (AAB) rather than APK formats. This mechanism mandates that developers share the private signing key so that Google can generate the app bundle and sign the AAB with the same private key [81]. With this access, Google has the privilege to conduct an integrity check to support its app security improvement program and remediation.

Another recent major update was conducted in June 2022, when Google changed the security and privacy policies and drastically changed the interface. The changes force the app's developers to align their SDKs with Google Play Store policies. This mechanism allowed Google to control and capture usage statistics, crash reporting, and give the ability for SDK providers to communicate with app developers through Play Console and Android Studio. Another security measure by Google comes from improving Play's app-integrity tools. Google Play App Signing helps millions of apps on Google Play ensure that app updates can be trusted [52].

After all these security system improvements, we then retraced the existence of malicious mHealth on the Google Play Store. Out of a total of 381 malicious mHealth that we have, we found that 188 malicious apps are still on the Play Store. Then, to see the existence of malicious mHealth apps, we downloaded 188 apps and evaluated them against VirusTotal. As a result, 88 mHealth apps were detected as containing malware, with 16 having an AV-rank of 5 or above. This indicates that the security improvements of the Google Play Store did not detect and remove suspicious mHealth apps from their repository. Table 8 shows the Top-10 malicious mHealth currently on the Google Play Store.

Table 7: Sha256 hash comparison among 381 mHealth apps, 188 mHealth that still (as of August 31, 2022) exist at Google Play.

Suspicious,	Apps in 2022			
Found in 2021	Found	Updated	Suspicious	# of Apps(%)
✓	-	-	-	381 (100%)
✓	✗	-	-	193 (50.7%)
✓	✓	-	-	188 (49.3%)
✓	✓	✓	-	140 (36.7%)
✓	✓	✗	✓	37 (19.7%)
✓	✓	✓	✓	51 (27.1%)
✓	✓	✓	✗	89 (47.3%)

To observe if the malicious behaviour was obtained from the prolonged mHealth apps, we then compared the Hashed files of the mHealth apps with mHealth that existed after June 2022 and also with mHealth that was still detected as malicious by VirusTotal after June 2022. Table 7 lists that 140 of the 188 apps are new apps and the remaining 48 apps are apps that have not been updated. Even though they have been updated, 51 mHealth apps still inherit malicious behaviour from the previous version, while the remaining 89 have been whitelisted by VirusTotal.

User Awareness Analysis of the user awareness with respect to the presence of malware in the apps found that only 4% (15) of the apps have negative reviews, which correlates to the type of malware present in the apps. In 14 of these apps, the negative reviews relate to excessive advertising hindering the functionality of the app. Only one app has a review recognising a Trojan found through the use of antivirus software.

82% of the apps have more than 1K installs. Six apps have 1M+ installs and nine have 500K+. All 15 apps with more than 500K installs have an $AV\text{-}rank \leq 3$. Eight of the apps that have 100K+ installs have an $AV\text{-}rank \geq 4$, with two having an $AV\text{-}rank \geq 20$. Figure 4 shows that the majority of apps are positively rated by users, with 54% (204) of the apps having a rating > 3.0.

Table 8: List of suspicious mHealth apps that still accessible (at time of writing, August 21, 2022) on Google Play. Here AV-rank shows the number of antivirus tools agreeing on flagging an mHealth app as suspicious.

No	App Title	AV-rank	#of Install
1	Fit Bites [67]	17	10,000+
2	Smoke'n Vap'z [11]	17	10+
3	Vap'Pause [13]	16	100+
4	Spa Thérapie [12]	16	10+
5	Cellu hit [9]	16	10+
6	Institut O'plaisir [8]	15	50+
7	COMM [7]	15	100+
8	Calorie Calculator BMR BMI ads [38]	14	1,000+
9	OPTIKONCEPT [10]	13	10+
10	BMI Calculator [37]	13	1,000+
11	(agricultural, food, safety) [51]	12	1,000+
12	TeleEmergencias [59]	12	100+
13	Cara Gaya Renang [16]	7	1,000+
14	7 Minute Super Plank Workout [75]	6	50,000+
15	1byone Health [2]	6	100,000+
16	Cara Menghilangkan Jerawat [18]	5	500+
17	Learn To Do Pull-Ups [74]	4	50,000+
18	simvalley PhoneWatch [72]	3	100,000+
19	Cara Latihan Badminton [17]	3	10,000+
20	BodyMonitor [88]	2	100,000+

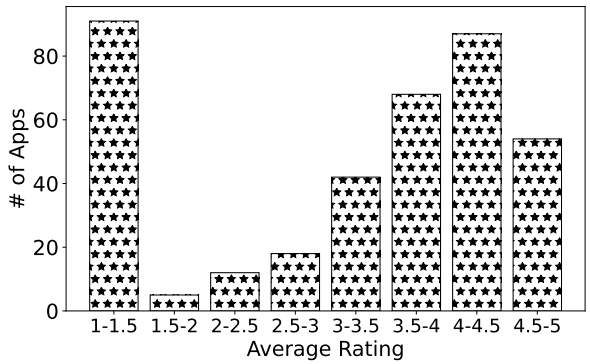

Figure 4: Distribution of average ratings of *Suspicious* mHealth apps at Google Play.

7 RELATED WORKS

Some research related to mobile health apps security and privacy analysis has been done previously, including a study conducted by [26], that focuses on the establishment of an overview of mHealth apps offered on iOS and Android with a particular focus on potential damage to users through information security and privacy infringements. [61] conducted research that focused on identifying relevant security concerns on the server side of mHealth apps and comparing the servers used by mHealth apps with servers used in all domains. [87] conducted the first large-scale analysis of mobile health (mHealth) apps available on Google Play to provide a comprehensive view of mHealth apps' security features and gauge the associated risks for mHealth users and their data. While [86] investigated whether and what user data is collected by mHealth apps, characterised the privacy conduct of all the available mHealth apps on Google Play, and gauged the associated risks to privacy. [4] focuses on reviewing and analysing privacy policies, data sharing, and security policies of women's mHealth apps on the Apple Store and Google Play. [80] researched security and privacy analysis of apps that provide tracking and checking symptoms of health conditions or diseases through mobile devices. However, none of the mentioned research on the characterization and attribution of malicious behaviour appears on mobile health apps, which become our focus in this study.

8 DISCUSSION AND RESPONSIBLE DISCLOSURE

Several studies have conducted security and privacy measurements of mHealth apps using standard parameters. However, none of these studies conducted an audit and attribution of the causes of maliciousness from mHealth apps. Our interesting finding begins when we characterize the malware found by VIRUSTOTAL, where there is a similarity in name between the malware family and the commercial advertising library that is circulated widely. Based on this fact, we observed further and found that 19.7% of the suspicious mHealth developers had abused the advertising library to gain privilege and massively invoked advertising content in order to monetize their apps. This coherent is considering that all of the mHealth apps in our corpus are free and require income for maintenance.

We also found a pattern in packages and directories naming based on the third-party libraries' analysis. Our further research found that 29.9% suspicious mHealth were built using the Online App Generator (OAG). This framework is a medium for suspicious mHealth app developers to insert malicious code, which is only a small proportion of the total code. This is supported by the results of the entropy measurement at the byte level code, where we did not find a significant delta among mHealth apps' entropy that embeds different binary protectors. This small entropy is caused by a small delta of code, considering that the malicious code is only a small part of the total APK package.

Our analysis of API calls and runtime found that there was a massive user information invocation being sent to a third-party server. We also found that 20 mHealth apps communicated with domains flagged as malicious. However, what is worrisome is the user's low level of awareness of this information harvesting. This is indicated by a rating that has an average of above three, which shows that the user still has a good perspective on this suspicious mHealth. We suspect that users are unaware that their data has been harvested by a third party or do not really care if the data is sent to a third party. From a developer's perspective, we also suspect that some developers are unaware that their application is detected as malicious, considering that many apps are developed with OAG and embed third-party libraries. Therefore, we believe this study's results can provide a new perspective on the severe problems that suspicious mHealth poses for users and developers.

Responsible Disclosure. We contacted and shared our findings with the app developers of apps requesting sensitive permissions, apps that users negatively review, and apps with embedded third-party tracking libraries. We also contacted app developers, which our tests revealed as possibly containing malware in their APKs. We have disclosed and confirmed our findings to the mHealth

developers mentioned in this paper. We have not yet received any responses from the contacted developers. We will update our paper according to the responses and the confirmations of the findings from the developers.

9 CONCLUSION

Despite being better aligned with security best practises than non-mHealth apps, suspicious mHealth apps are still accessible on Google Play and can potentially expose consumers to a wide range of security issues. App users, clinicians, technology developers, and policy-makers alike should be cognizant of the uncovered security issues and weigh them carefully against the benefits of mHealth apps. To this end, we performed the first comprehensive and in-depth analysis of mechanisms used by suspicious mHealth apps. This is the first study to conduct characterization and attribution of the mHealth apps' malicious behavior on the Google Play Store. The study revealed several types of malicious apps, including those that massively collect and share user information with third-party servers; apps that use modified and malicious libraries; and apps that embed libraries that illegally harvest user information from legit libraries. We believe that the toolset developed in this, in particular, the extraction of EMBER-like features, is a promising alternative way to complement the literature in analyzing Android applications. We aim to release our advanced toolset upon publication and share data with the research community.

REFERENCES

[1] [n.d.]. LIEF Documentation - Android formats and the API to use them. https://lief-project.github.io/doc/latest/tutorials/10_android_formats.html.
[2] 1byone. [n.d.]. 1byone Health. https://play.google.com/store/apps/details?id=com.quhwa.health.
[3] Airnow. [n.d.]. Airnow. https://airnowmonetization.com
[4] Najd Alfawzan, Markus Christen, Giovanni Spitale, and Nikola Biller-Andorno. 2022. Privacy, Data Sharing, and Data Security Policies of Women's mHealth Apps: Scoping Review and Content Analysis. *JMIR Mhealth Uhealth* 10, 5 (6 May 2022).
[5] Hyrum S Anderson and Phil Roth. 2018. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).
[6] Apktool. [n.d.]. Apktool. https://ibotpeaches.github.io/Apktool/
[7] AppsVision. [n.d.]. COMM. https://play.google.com/store/apps/details?id=com.appsvision.comm.
[8] AppsVision. [n.d.]. Institut O'plaisir. https://play.google.com/store/apps/details?id=com.appsvision.institutoplaisir
[9] AppsVision1. [n.d.]. Cellu hit. https://play.google.com/store/apps/details?id=com.appsvision.celluhit
[10] AppsVision3. [n.d.]. OPTIKONCEPT. https://play.google.com/store/apps/details?id=com.appsvision.optikoncept
[11] AppsVision4. [n.d.]. Smoke'n Vap'z. https://play.google.com/store/apps/details?id=com.appsvision.smokenvapz
[12] AppsVision5. [n.d.]. Spa Thérapie. https://play.google.com/store/apps/details?id=com.appsvision.spatherapie
[13] AppsVision6. [n.d.]. Vap'Pause. https://play.google.com/store/apps/details?id=com.appsvision.vappause
[14] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.
[15] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.
[16] AttenTS. [n.d.]. Cara Gaya Renang. https://play.google.com/store/apps/details?id=com.caragayarenang.atten.
[17] AttenTS. [n.d.]. Cara Latihan Badminton. https://play.google.com/store/apps/details?id=com.caralatihanbadminton.atten.
[18] AttenTS. [n.d.]. Cara Menghilangkan Jerawat. https://play.google.com/store/apps/details?id=com.caramenghilangkanjerawat.atten.
[19] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 217–228.
[20] Maged N Kamel Boulos, Ann C Brewer, Chante Karimkhani, David B Buller, and Robert P Dellavalle. 2014. Mobile medical and health apps: state of the art, concerns, regulatory control and certification. *Online J. Public Health Inform.* 5, 3 (2014).
[21] BouncyCastle. [n.d.]. Raccoon – The APK Donwloader. https://raccoon.onyxbits.de/
[22] Sun Caijun, Zhang Hua, Qin Sujuan, He Nengqiang, Qin Jiawei, and Pan Hongwei. 2018. DexX: A Double Layer Unpacking Framework for Android. *IEEE Access* (2018).
[23] N. Chau and S. Jung. 2019. An Entropy-Based Solution for Identifying Android Packers. *IEEE Access* (2019).
[24] Caleb Chen. 2020. Android community worried about presence of "Chinese spyware" by Qihoo 360 in Samsung smartphones and tablets. https://www.privateinternetaccess.com/blog/android-community-worried-about-presence-of-chinese-spyware-by-qihoo-360-in-samsung-smartphones-and-tablets/
[25] Nathan Collier. 2021. Pre-installed auto installer threat found on Android mobile devices in Germany. https://www.malwarebytes.com/blog/news/2021/04/pre-installed-auto-installer-threat-found-on-android-mobile-devices-in-germany. Last accessed: 10/08/2022.
[26] Tobias Dehling, Fangjian Gao, Stephan Schneider, and Ali Sunyaev. 2015. Exploring the Far Side of Mobile Health: Information Security and Privacy of Mobile Health Apps on iOS and Android. *JMIR Mhealth Uhealth* 3, 1 (Jan. 2015), e8.
[27] Android Developer. 2020. *The Android NDK: toolset that lets you implement parts of your app in native code, using languages such as C and C++.*
[28] Digital.ai. 2020. Arxan: App code obfuscation. https://digital.ai/glossary/app-code-obfuscation. Last accessed: 18/08/2021.
[29] Yue Duan, Mu Zhang, Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li, Xueqiang Wang, and XiaoFeng Wang. 2018. Things You May Not Know About Android (Un)Packers: A Systematic Study based on Whole-System Emulation. In *NDSS*.
[30] Nguyen Viet Duc, Pham Thanh Giang, and Pham Minh Vi. 2015. Permission Analysis for Android Malware Detection. In *The Proceedings of the 7th VAST-AIST Workshop "Research Collaboration: Review and perspective.*
[31] F-Secure. 2017. F-Secure -Threat Description - Riskware:Android/SmsReg. https://www.f-secure.com/sw-desc/riskware_android_smsreg.shtml.
[32] Caleb Fenton. 2016. *Building with and Detecting Android's Jack Compiler.*
[33] Fmind. [n.d.]. fmind/euphony. https://github.com/fmind/euphony
[34] US Food and Drug Administration. 2019. Policy for Device Software Functions and Mobile Medical Applications. https://www.fda.gov/media/80958/download
[35] Fortiguard. 2017. Fortiguard labs - Threat Encyclopedia - Android/SMSreg.ZI!tr. https://www.fortiguard.com/encyclopedia/mobile/7294379/android-smsreg-zi-tr.
[36] Fortinet. 2021. Fortiguard labs - Threat Encyclopedia - Adware/RevMob. https://www.fortiguard.com/encyclopedia/mobile/7016460/adware-revmob. Last accessed: 10/08/2022.
[37] Marco Grewenig. [n.d.]. BMI Calculator. https://play.google.com/store/apps/details?id=de.grewe.android.bmi.
[38] Marco Grewenig. [n.d.]. Calorie Calculator BMR BMI ads. https://play.google.com/store/apps/details?id=de.grewe.android.caloriecalculatorfree.
[39] Guardsquare-Mobile-Application-Protection. 2020. *Dexguard: Android App Security - Protecting Android applications and SDKs against reverse engineering and hacking.*
[40] Ren He, Haoyu Wang, Pengcheng Xia, Liu Wang, Yuanchun Li, Lei Wu, Yajin Zhou, Xiapu Luo, Yao Guo, and Guoai Xu. 2020. Beyond the Virus: A First Look at Coronavirus-themed Mobile Malware. arXiv:2005.14619 [cs.CR]
[41] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. 2017. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for Android malware. In *MSR*.
[42] Muhammad Ikram and Mohamed Ali Kaafar. 2017. A first look at mobile ad-blocking apps. In *NCA*. IEEE, 1–8.
[43] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. 2019. The chain of implicit trust: An analysis of the web third-party resources loading. In *The World Wide Web Conference*. 2851–2857.
[44] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. 2016. An analysis of the privacy and security risks of android VPN permission-enabled apps. In *Proceedings of the 2016 Internet Measurement Conference*. 349–364.
[45] Mansoor Iqbal. [n.d.]. App download and usage statistics. https://www.businessofapps.com/data/app-statistics/
[46] Jean-Yves Jenny. 2013. Measurement of the knee flexion angle with a Smartphone-application is precise and accurate. *J. Arthroplasty* 28, 5 (May 2013), 784–787.

[47] Ryan Johnson, Zhaohui Wang, Corey Gagnon, and Angelos Stavrou. 2012. Analysis of android applications' permissions. In *2012 IEEE Sixth International Conference on Software Security and Reliability Companion*. IEEE, 45–46.

[48] Misha Kay, Jonathan Santos, and Marina Takane. 2011. mHealth: New horizons for health through mobile technologies. *World Health Organization* 64, 7 (2011).

[49] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. Cutting the gordian knot: A look under the hood of ransomware attacks. In *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 3–24.

[50] Leadbolt. 2021. Leadbolt - High Performance Mobile Advertising. https://www.leadboltapps.com/. Last accessed: 10/08/2022.

[51] Jiunn-Jyh Lin. [n.d.]. Production and sales history traceability number query (agricultural products, food), safety traceability, production traceability. https://play.google.com/store/apps/details?id=com.xcodeon.android.foodtraceability

[52] Iryna Lukashuk. [n.d.]. Google Play Store changes 2022: what to expect? https://appradar.com/blog/google-play-store-changes-2022 published : June 14, 2022 , last access: August 2022.

[53] Malicialab. 2020. malicialab/avclass. https://github.com/malicialab/avclass

[54] Sapna Malik and Kiran Khatter. 2016. Behaviour Analysis of Android Application. *IJCTA* 9, 10 (2016).

[55] Malwarebyte. 2019. Malwarebyte Forum - Android/PUP.Riskware.SMSreg.WWPA. https://forums.malwarebytes.com/topic/249727-androidpupriskwaresmsregwwpa/.

[56] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 55–60.

[57] Microsoft. 2021. Microsoft Security Intelligence - PUA:AndroidOS/SMSReg.I!MTB. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PUA:AndroidOS/SMSReg.I!MTB&threatId=324027.

[58] mitmproxy. [n.d.]. https://mitmproxy.org

[59] molinoapp. [n.d.]. TeleEmergencias. https://play.google.com/store/apps/details?id=com.ApkCGL.teleemergencia.

[60] MonkeyRunner. [n.d.]. monkeyrunner: Android Developers. https://developer.android.com/studio/test/monkeyrunner

[61] Jannis Müthing, Raphael Brüngel, and Christoph M Friedrich. 2019. Server-Focused Security Assessment of Mobile Health Apps for Popular Mobile Platforms. *J Med Internet Res* 21, 1 (23 Jan 2019).

[62] Rahul Nair. 2015. *Techbliss - Tutorial Anti-Disassembly techniques used by malware (a primer)*.

[63] Andréa A G Nes, Sandra van Dulmen, Erlend Eide, Arnstein Finset, Olöf Birna Kristjánsdóttir, Ida Synnøve Steen, and Hilde Eide. 2012. The development and feasibility of a web-based intervention with diaries and situational feedback via smartphone to support self-management in patients with diabetes type 2. *Diabetes Res. Clin. Pract.* 97, 3 (Sept. 2012), 385–393.

[64] Ammar Odeh, Ismail Keshta, Abobakr Aboshgifa, and Eman Abdelfattah. 2022. Privacy and Security in Mobile Health Technologies: Challenges and Concerns. In *CCWC*.

[65] Marten Oltrogge, Erik Derr, Christian Stransky, Yasemin Acar, Sascha Fahl, Christian Rossow, Giancarlo Pellegrino, Sven Bugiel, and Michael Backes. 2018. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. In *2018 IEEE Symposium on Security and Privacy (SP)*.

[66] S O'Neill and R R W Brady. 2012. Colorectal smartphone apps: opportunities and risks. *Colorectal Dis.* 14, 9 (Sept. 2012), e530–4.

[67] D Orange. [n.d.]. Fit Bites. https://play.google.com/store/apps/details?id=com.fit.bites

[68] OWASP. 2020. *Testing Anti-Debugging Detection (MSTG-RESILIENCE-2) - Android Anti-Reversing Defenses*. OWASP Mobile Security Guide - Accessed: 18/01/2020.

[69] OWASP. 2020. *Testing Emulator Detection (MSTG-RESILIENCE-5) - Android Anti-Reversing Defenses*. OWASP Mobile Security Guide - Accessed: 18/01/2020.

[70] Achilleas Papageorgiou, Michael Strigkos, Eugenia Politou, Efthimios Alepis, Agusti Solanas, and Constantinos Patsakis. 2018. Security and Privacy Analysis of Mobile Health Applications: The Alarming State of Practice. *IEEE Access* 6 (2018).

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

[72] PEARL.GmbH. [n.d.]. simvalley PhoneWatch. https://play.google.com/store/apps/details?id=de.pearl.px4555.

[73] Charlie Pinder, Jo Vermeulen, Benjamin R Cowan, and Russell Beale. 2018. Digital behaviour change interventions to break and form habits. *TOCHI* 25, 3 (2018).

[74] Body Program. [n.d.]. Learn To Do Pull-Ups. https://play.google.com/store/apps/details?id=bppullups.apps.com.

[75] Body Program. [n.d.]. Minute Super Plank Workout. https://play.google.com/store/apps/details?id=bpsplankw.apps.com.

[76] T E Schap, F Zhu, E J Delp, and C J Boushey. 2014. Merging dietary assessment with the adolescent lifestyle. *J. Hum. Nutr. Diet.* 27 Suppl 1 (2014), 82–88.

[77] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. Avclass: A tool for massive malware labeling. In *RAID*.

[78] Rednaga Security. 2016. *APKiD and Android Compiler Fingerprinting*.

[79] Rednaga Security. 2016. *Detecting Pirated and Malicious Android Apps with APKiD*.

[80] I. Wayan Budi Sentana., Muhammad Ikram., Mohamed Kaafar., and Shlomo Berkovsky. 2021. Empirical Security and Privacy Analysis of Mobile Symptom Checking Apps on Google Play. In *Proceedings of the 18th International Conference on Security and Cryptography - SECRYPT,*.

[81] Arjun Sha. [n.d.]. APK vs AAB (Android App Bundles): Everything You Need to Know! https://beebom.com/apk-vs-aab/ published : July 5, 2021 , last access: August 2022.

[82] SIMFORM. 2015. *How to avoid reverse engineering of your android app?*

[83] David Smahel, Steriani Elavsky, and Hana Machackova. 2019. Functions of mHealth applications: A user's perspective. *HIJ* 25, 3 (2019).

[84] Google Source. 2020. Android Clang/LLVM Prebuilts. https://android.googlesource.com/platform/prebuilts/clang/host/linux-x86/+/master/README.md. Last accessed: 18/08/2021.

[85] Soylentnews.org. 2020. Samsung Devices Allegedly Use Qihoo 360 Spyware to Phone Home to China. https://www.newsbreak.com/news/1486552211048/samsung-devices-allegedly-use-qihoo-360-spyware-to-phone-home-to-china

[86] Gioacchino Tangari, Muhammad Ikram, Kiran Ijaz, Mohamed Ali Kaafar, and Shlomo Berkovsky. 2021. Mobile health and privacy: cross sectional study. *British Medical Journal* 373 (June 2021), n1248.

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

[88] Senssun Technologies. [n.d.]. BodyMonitor. https://play.google.com/store/apps/details?id=com.senssun.bodymonitor.

[89] Kevin Townsend. 2020. Threat From Pre-Installed Malware on Android Phones is Growing. https://www.securityweek.com/threat-pre-installed-malware-android-phones-growing. Last accessed: 10/08/2022.

[90] VirusTotal. [n.d.]. https://www.virustotal.com/gui/home/upload

[91] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, and Yuqing Zhang. 2021. Understanding Malicious Cross-library Data Harvesting on Android. In *USENIX Security Symposium*.

[92] Dr Web. 2021. Adware.Leadbolt.24 Technical Information. https://vms.drweb.com/virus/?i=24980493&lng=en. Last accessed: 10/08/2022.

[93] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*. IEEE, 95–109.

A EVASIVE AND OBFUSCATION METHODS

In this section, we provide details about the six different types of evasion techniques employed by mHealth apps. For each application, we use `APKiD`[2] tool to obtain a list anti-analysis techniques such as "manipulator", "anti-virtual machine", "anti-debug", "anti-disassembly", "obfuscator", and "packer".

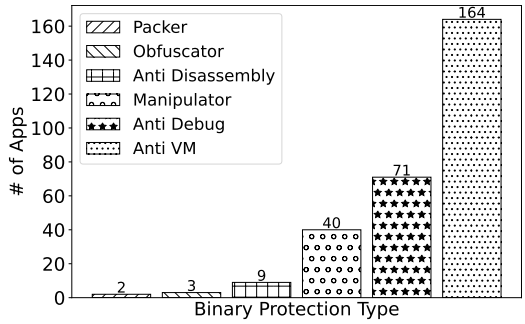

Figure 5: Distribution of protection and hiding mechanims employed by the analysed suspicious mHealth apps.

Manipulator. We found that 40 (10.5%) apps are marked as containing manipulator because the *Dalvik Executable (.dex)* files

[2]https://github.com/rednaga/APKiD

developed using *dexmerge* compiler. Note that *.dex* file, which exists in each APK, is a byte-code file converted from *Java.class*. Thus, it can be executed by the devices. Originally, *dex* file was developed using *dx* or *r8* compiler. APKiD tool will mark an app as containing manipulator if (i) the original *.dex* files of the app are modified using a modification library such as *dexmerge* or (ii) *.dex* files are created from reverse-engineered source code using *dexlib* library, which is commonly used by decompiler tools such as apktool or smali [78, 79]. APKiD tool identifies the manipulator by analyzing the change history in *Map Ordering Type* of the *.dex* files since the code sequence resulted from original *.dex* compiler, *dexmerge, dexlib or dex2lib* is different [32].

Anti Virtual Machine. We detected 164 (43.04%) apps adopting anti-virtual machine (anti-vm) analysis in their packages. Anti-vm is a mechanism to detect whether the apps are executed on an emulator or a real device. The goal is to impede reverse-engineering tools and techniques so the reverse engineer cannot get the source code easily. The most common mechanism to check whether the device is emulated [69] is to analyze *build.prop* file containing a list of Build API methods, including *Build.Fingerprint, Build.Hardward, Build.Device* and other device's properties. An alternative method is to check the *Telephony manager* which contains fixed API values for Android emulators including `getNetworkType()`, `getNetworkOperator()`, `getPhoneType()` and other network-related properties.

Anti Debug. We found 71 (18.63%) apps leveraging anti-debugging techniques to disrupt reverse engineering of their source code. Anti-debugging technique ensures that the apps do not run under a debugger or change the app's behaviour when running under the debugger mode. Android provides two levels of debugging and anti-debugging protocol [68]. The first debugging level can be conducted in communication protocol between Java Virtual Machine and debugger using Java Debug Wire Protocol (JDWP). We can identify anti-debugging by verifying if the setup includes the *debuggable flag* in *ApplicationInfo* or by checking the *timer checks routine*. In contrast, the next level of anti-debugging technique is to conduct traditional debugging by using *ptrace* in Linux *system call*. In this research, we found all of the mHealth apps activating the debugable flag of `Debug.isDebuggerConnected() check`, which is part of the JDWP anti-debug level.

Obfuscator. An app developer commonly uses an obfuscator to protect intellectual property or trade secrets and prevent an attacker from reverse engineering a proprietary software program by encrypting some or all of the program's code. There are several popular obfuscator tools, including Dexguard [39], Arxan [28], and Clang [84]. Dexguard is a proprietary Android obfuscation tool that provides multi-layer protection against the static and dynamic analysis of byte-code, manifest, and all other resources included in distribution packages. While Dexguard obfuscated the byte-code level of Android Apps, Arxan and Clang are categorized as Low-Level Virtual Machine tools that obfuscate the binary code level of Android Apps. In this study, we found that APKID detected 3 (0.78%) apps leveraging obfuscator tools, where 2 of them were obfuscated using Low-Level Virtual Machine Tools (LLVM) such as Clang, and the rest are unidentified.

Anti Disassembly. This technique prevents the reverse engineer from disassembling the byte-code into higher-level code such as Java or Smali. The most popular anti-disassembly mechanism in Android is by developing part of the code segment in C or C++ using the Native Development Kit (NDK) [82]. NDK provides platform libraries to manage native activities and access physical device components [27]. NDK uses *CMake* as a native library compiler that creates a different *byte-code* structure compared to the code written in Java or Kotlin. Hence, it impedes common Android tools such as Apktool or Smali from disassembling the *byte-code*. It required an advanced reverse engineer familiar with the ARM processor architecture, Assembler language, Java Native Interface (JNI) convention, and Application Binary Interface (ABI) compiler to decompile the *byte-code*. Malware developers use more advanced techniques to evade disassembly tools, as explained by [62]. The technique is leveraging *jmp* and *call* commands in *byte-code* level to direct the instruction flow to a particular location with a constant value or direct the flow to the exact target memory location. This technique will produce an incorrect source code listing when it is disassembled using a decompiler tool. We found 9 (2.3%) apps leveraging the anti-disassembly techniques. Analysis of those apps returns the value "Illegal class name", indicating the decompiler result violates the standard structure of Java or Kotlin.

Packer. Initially, Packer was created to protect intellectual property in the form of source code on Android apps. These tools prevent third parties from analyzing source code or doing reverse engineering. Packer works by encrypting the `.dex` files in the Android package and storing the encryption results in a secure new block architecture. Unlike the obfuscator, which will be decrypted when executed on the device, the packer remains stored in the packer block and uses unpacker when executed on the device. However, malware developers often use commercial packers to hide malicious codes [22, 29]. Research in [29] shows that several commercial packers are abused to encrypt malicious code in apps. Research on [22] also explains how packers are used to deceive Google Play Store scans so that reproducible malware called *Expensive Wall* is embedded in one of the apps in the apps market. In this research, we found that 2 (0.5%) mHealth apps leverage "Jiagu" packer.

Based on the analysis result and considering the malware self-protection behaviour claim in [40], we believe that mHealth suspicious apps have different behaviour. Except for anti-virtual machines, suspicious mHealth apps adopting other binary protector mechanisms have less than 50%. Implementing anti-VM is straightforward since several Software Development Kits (SDK) have included it in their systems. But adopting the other mechanism, such as obfuscation, might take more effort and investment. Since the mHealth apps are not considered potentially profitable for conducting a crime, we suspect that the app's developer exploits the mHealth apps to launch more ad-related libraries. That is aligned with the type of malicious library detected by Virus Total in Table 2.

As a note, we believe several results provided by APKId, such as the obfuscation mechanism, are considered to be the lower-bound results because, based on our static analysis, we found at least 16 apps leveraging the Airpush and eight apps leveraging the LeadBolt library (see Table 4) are obfuscated by renaming the library's name. This is because APKId conducted fingerprinting based on a popular obfuscator whitelist and byte-level analysis, which can be easily missed through simple renaming techniques.