Journal of Internet Services and Applications, 2024, 15:1, doi: 10.5753/jisa.2024.3882

COVID-19 Mobile Applications: A Study of Trackers and Data Leaks

Nicolás Serrano [**Pedeciba Informática** | ***nicolas.serrano@fing.edu.uy***]
Gustavo Betarte [**Universidad de la República** | ***gustun@fing.edu.uy***]
Juan Diego Campo [**Universidad de la República** | ***jdcampo@fing.edu.uy***]

Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, Montevideo, Uruguay.

Received: 28 November 2023 • **Accepted:** 10 April 2024 • **Published:** 24 July 2024

Abstract The emergence of COVID-19 in 2019 had a profound international impact. Technologically, governments and significant organizations responded by spearheading the development of mobile applications to aid citizens in navigating the challenges posed by the pandemic. While many of these applications proved successful in their intended purpose, the safeguarding of user privacy was not consistently prioritized, revealing a prevalent use of third-party libraries commonly referred to as trackers. In our comprehensive analysis encompassing 595 Android applications, we uncovered trackers in 402 of them, leading to the inadvertent exposure of sensitive user information and device data on external servers. Our investigation delved into the methodologies employed by these trackers to harvest and exfiltrate information. Furthermore, we examined the positions adopted by both trackers and governments. This study underscores the critical need for a reevaluation of the inclusion of trackers in applications of such sensitivity. Recognizing the potential lack of awareness within the scrutinized organizations regarding the risks associated with integrating third-party libraries, particularly trackers, we introduce SAPITO as part of our contributions. SAPITO is an open-source tool designed to identify potential leaks of sensitive data by third-party libraries in Android applications, providing a valuable resource for enhancing the security and privacy measures of mobile applications in the face of evolving technological challenges.

Keywords: Mobile apps, data leaks, trackers, COVID-19

1 Introduction

Regrettably, an extensive preamble to the Coronavirus (COVID-19) is unnecessary at this juncture. This infectious disease, stemming from the SARS-CoV-2 virus, manifests in individuals with mild to moderate respiratory symptoms [World Health Organization, 2020]. Initially reported in late 2019, COVID-19 swiftly evolved into a pervasive global pandemic. As of the current writing, nearly 800 million infections and approximately 7 million fatalities have been attributed to this virus. Although the weekly mortality rate has diminished compared to the initial two years of the pandemic, it remains a significant and pressing global health concern.

Technology has seamlessly integrated into every facet of our daily existence. A domain witnessing consistent growth in recent years is digital health, encompassing mobile health, health information technology, wearable devices, telehealth, telemedicine, and personalized medicine. Against the backdrop of the COVID-19 pandemic, digital health mobile applications (hereafter referred to as apps) became ubiquitous across nearly every country. Governments, international organizations, health institutions, and universities were among the key sponsors propelling the development of these technologies. This study directs its focus towards the examination of mobile apps specifically developed, provided, sponsored, or funded by these entities, with a selection of examples outlined in Table 1.

The exploration of this specific category of apps is motivated by the inherent trust vested in the institutions sponsoring them by their intended users. Notably, governmental apps, in certain instances, became obligatory or expedient for individuals to seamlessly engage in their daily activities—such as when presenting vaccination certificates stored in the application to access public spaces. Amid the COVID-19 landscape, data protection and privacy emerge as paramount concerns, particularly in apps designed for contact tracing. Embracing a ”privacy by design” ethos is imperative, demanding meticulous adherence to regulations such as HIPAA [US Department of Health and Human Services, 1996], GDPR [European Parliament, 2016], and other pertinent data protection regulations. The integration of data protection impact assessments becomes indispensable within the developmental life cycle of applications of this scale. Equally critical are comprehensive privacy policies ensuring that users are well-informed and willingly consent to the utilization of their data by the application, its collectors, and processors. Any data leakage (when sensitive information is unintentionally exposed beyond its authorized scope) within the context of these apps has the potential to undermine the effectiveness of contact-tracing efforts conducted by countries. From a technical standpoint, the decision of some developers to release the application code as open-source stands out as a commendable practice fostering transparency. The relation between transparency and privacy is crucial, and their balance is essential to cultivate user trust in the evolving land-

Country	Name	Developer	Goal
Australia	Coronavirus Australia	Australian Department of Health	Official Information
Brazil	Coronavírus - SUS	Ministério da Saúde	COVID Guidance (*)
Denmark	Coronapas	Danish Ministry of Health	COVID Passport
France	TousAntiCovid	Ministère de la Santé et de la Prévention	Contact Tracing (*)
Germany	Corona-Datenspende	Robert Koch-Institut	Studies
Hong Kong	StayHomeSafe	Office of the Gvment. Chief Inf. Officer	Quarantine Enforcement
India	West Bengal Emergency Fund	Govt. of West Bengal	Donations
International	COVID-19: response	United Nations	Information
Italy	LAZIOdrCovid	Salute Lazio	Telemedicine
Jordan	Cradar	Nat. Center for Sec. and Crisis Mgment.	Gatherings Denouncing
Malaysia	MySejahtera	Government of Malaysia	Self-Diagnostic (*)
New Zealand	Āwhina - for health workers	Ministry of Health	Health Workers Support
Norway	Kontroll av koronasertifikat	Institute of Public Health	Certificates Verification
UAE	COVID-19 EHS	Emirates Health Services	Vaccination (*)
Uruguay	Coronavirus UY	E-Government Agency	Statistics (*)
US	COVID Coach	US Department of Veterans Affairs	General Well-Being
Vietnam	Vietnam Health Declaration	Ministry of Health	Travelers Health Declaration

Table 1. Example of analyzed apps and their main purposes. (*) These apps have other purposes besides those stated in the table.

scape of digital health applications.

In contemporary software development, it is commonplace for mobile applications to integrate third-party libraries using software development kits (SDKs). This practice is widely adopted owing to the manifold benefits these libraries confer upon developers, encompassing specialized functions such as map integration, login with social media credentials, monetization features, crash reporting, and user engagement enhancements. However, it is crucial to note that certain libraries go beyond their apparent functionalities, actively collecting sensitive information pertaining to users, their devices, and the interaction between the two. The spectrum of data tracked spans diverse categories, including but not limited to *location*, *device information*, *application attributes*, and *battery status*. This amassed data serves for constructing user profiles, subsequently leveraged for purposes such as targeted advertising.

While tracker libraries covet any user information they can access, the custodian of COVID-19-related sensitive data shoulders a profound responsibility. Consequently, the collection and dissemination of data to third parties must be circumscribed to the essential minimum and strictly aligned with the specific objectives of the application. This imperative resonates with the approach adopted by many countries in the development of their contact tracing apps, as articulated in [ICO United Kingdom, 2020]: *"13. It is important to clearly define the use of coding libraries, frameworks, APIs, SDKs, and other software components, including those within the mobile operating system. To avoid data collection by third parties for unrelated purposes."*

In the context of this study, we define *trackers* as third-party libraries incorporated into applications to offer specific functionalities, which, additionally, collect information about the application, the device, and their usage, subsequently transmitting this data to their respective servers.

Research Questions. We conducted a meticulous examination of 595 COVID-19-related Android apps sourced from reputable platforms to ascertain the presence of trackers. Our analysis delved into the data collected and evaluated its repercussions on users. This study addresses the following inquiries:
(RQ1) Are trackers being used in the COVID-19 app ecosystem? How did this usage evolve over time?
(RQ2) What information is tracked and how?
(RQ3) What are the potential impacts on users if trackers are in use and harvest that information?

Contributions. Our investigation centered on assessing the impact of trackers within COVID-19 apps, a dimension largely unexplored in previous studies. Our discoveries reveal a substantial exchange of sensitive information with the servers of most trackers, encompassing data that could potentially unveil health and COVID-19-related details of the user. In contrast to prior studies, our research encompassed a comprehensive analysis of 595 Android apps, representing a significantly larger sample size. This breadth ensures that our findings offer a representative overview of COVID-19 apps, particularly those endorsed by governments and high-profile organizations. Furthermore, we expound on various privacy-centric initiatives undertaken by governments, shedding light on disparities in app privacy policies and Data Protection Impact Assessments. Notably, our analysis underscores instances where the presence of trackers is either undisclosed or inadequately addressed in these critical documents. Finally, we encapsulate the knowledge garnered from our in-depth app analysis into SAPITO, a well-rounded tool poised to be instrumental in examining privacy leaks within Android apps arising from the integration of third-party libraries. Its holistic approach enables it to identify potential issues that might have been overlooked by other tools, thereby filling in gaps and offering a more comprehensive understanding of an app codebase.

The present paper constitutes a substantially extended and thoroughly revised version of [Serrano *et al.*, 2023c] by contributing the following:

1. the analysis and discussion, in section 4, of the harvesting methodologies used by the inspected trackers. We examine and put forward how these trackers collect data, when data collection happens, and the methods employed to transmit data to their servers;
2. the detailed presentation and description of the tool SAPITO in section 6

Structure of the paper. The rest of the paper is structured as follows: Section 2 discusses related work. In Section 3 we describe the methodology used in our investigation, and in Section 4, we put forward the results of our research, provide details of the analyzed trackers and discuss the privacy issues caused by those tools. Section 5 elaborates on our findings and assesses their impact on violating data protection principles. In Section 6 we describe the developed tool. Section 7 concludes and discusses further work.

2 Related Work

We found plenty of research related to COVID-19 apps from different perspectives. For example: apps survey [Ahmed *et al.*, 2020], the platform governance [Dieter *et al.*, 2021], their effectiveness [Anglemyer *et al.*, 2020], their taxonomy [Samhi *et al.*, 2021], new approaches for contact tracing [Nakamoto *et al.*, 2020], COVID-19 themed malware [Wang *et al.*, 2021c], surveillance [Yang *et al.*, 2021], and overall lessons learned from the apps [Zhou *et al.*, 2021].

Academic research and industry studies on privacy within these apps have been conducted since the onset of the pandemic [Hatamian *et al.*, 2021; Wen *et al.*, 2020; Azad *et al.*, 2020; Ali *et al.*, 2020]. However, a majority did not center their focus on the impact of Software Development Kit (SDK) inclusion. This oversight could pose significant risks, as it potentially obscured a critical viewpoint on data privacy and the inadvertent sharing of personal information with third parties. Furthermore, an undue emphasis on application permissions was prevalent. While scrutinizing permissions is essential, it may not be inherently alarming for an application using *Bluetooth* for contact tracing or one facilitating video calls with a doctor to request *Camera* permissions.

Contrastingly, the scant studies that broached the subject of trackers predominantly limited their scope to merely identifying the included trackers [Kouliaridis *et al.*, 2021; AWO Agency, 2020]. In a parallel vein, [Dehaye and Reardon, 2020] demonstrated the ease with which third-party libraries can access private data through dynamic app analysis, linking it to the potential risks associated with contact tracing apps deployed for COVID-19 containment. Our study concurs with this conclusion, reached through a thorough static analysis of the tracker's code. Consequently, our in-depth exploration of Software Development Kits and their data tracking endeavors to fill the void left by previous studies, offering an indispensable perspective on privacy issues within COVID-19 apps.

Beyond the scope of COVID-19, the intersection of applications, trackers, and privacy has garnered attention in the research domain. Various tools dedicated to detecting privacy leaks at the application level have been introduced. For instance, [Continella *et al.*, 2017] presented *AGRIGENTO*, showcasing its superiority over previous tools. [He *et al.*, 2019] conducted dynamic privacy leakage analysis for third-party trackers. In our work, we opted for a broader approach, using static analysis of the code in the selected app and leveraging specific reverse engineering techniques and tools that are packaged in SAPITO. This decision was motivated by the dual objectives of precisely identifying the in-app library responsible for privacy breaches and minimizing the occurrence of false negatives. In this way, SAPITO supplements the existing tools by offering complementary insights and aiding in the thorough examination of apps.

The broader tracking landscape was explored by [Razaghpanah *et al.*, 2018] and [Binns *et al.*, 2018], providing insights into the ecosystem of tracking companies. Notably, the entities identified as predominant players in Advertising and Tracking Services (ATS) align with our research findings specifically focused on COVID-19 apps, with Alphabet, Facebook, and Microsoft featuring prominently in both contexts. Furthermore, our analysis corroborated the presence of additional ATS underscored in these comprehensive studies, including but not limited to AppsFlyer, AdColony, Adobe, Appnext, ComScore, Flurry, Lotame, MixPanel, New Relic, Segment, and Startapp.

[Liu *et al.*, 2020] enriched the understanding of privacy risks by concentrating on analytic libraries for Android, a departure from the predominant focus on advertisement libraries in existing research. Their findings demonstrated that apps also inadvertently disclose private information through these types of libraries. In alignment with this insight, our study analyzed 26 analytic libraries identified as trackers. Of these, the 10 subjected to further analysis to validate tracking behavior unequivocally exhibited the effective harvesting and exfiltration of user and mobile information.

The same was found in advertisement libraries for Android as reported in [Stevens *et al.*, 2012] and several other studies. We also found tracking behaviour in two out of three ad libraries flagged as trackers.

[Kollnig *et al.*, 2021] delved into the issue of user consent concerning data collection by third-party trackers. In our study, we identified instances where the companies responsible for trackers misrepresented the data collection in their declarations, with the actual data collected exceeding what was stated. Consequently, even if users were to provide potential consent, it would be incomplete for the actual data harvesting carried out by tracking libraries.

Recently, [Caputo *et al.*, 2022] not only delved into the ramifications of mobile analytic libraries but also introduced *MobHide*, a data anonymization tool crafted to address the potential vulnerabilities identified in our research, in case they surpass the current limitations of the tool. [Tangari *et al.*, 2021] conducted a targeted examination of mobile health apps, shedding light on the functionalities tracked by the embedded trackers. Drawing from our study's outcomes, we infer that COVID-19 apps, as a subset of mobile health apps, share the same vulnerability in safeguarding private information when shared with third parties. This holds true even though these apps, in general, receive backing from governmental and national agencies.

Examining the challenges faced by developers in deciding the convenience of using trackers, [Tahaei and Vaniea, 2021] provided insights into the unfavorable position developers find themselves in. Taking into account the urgency of launching COVID-19 apps amid a pandemic, we consider that development teams encountered similar dilemmas with the apps scrutinized in our study.

3 Methodology

This section delineates and deliberates on the methodological aspects that informed our research.

3.1 Tracker Detection

This section outlines the methodology we employed to identify trackers embedded in COVID-19-related apps and elucidates the process of analyzing the data they extract from the device, application, and user interactions. Figure 1 illustrates our methodology.

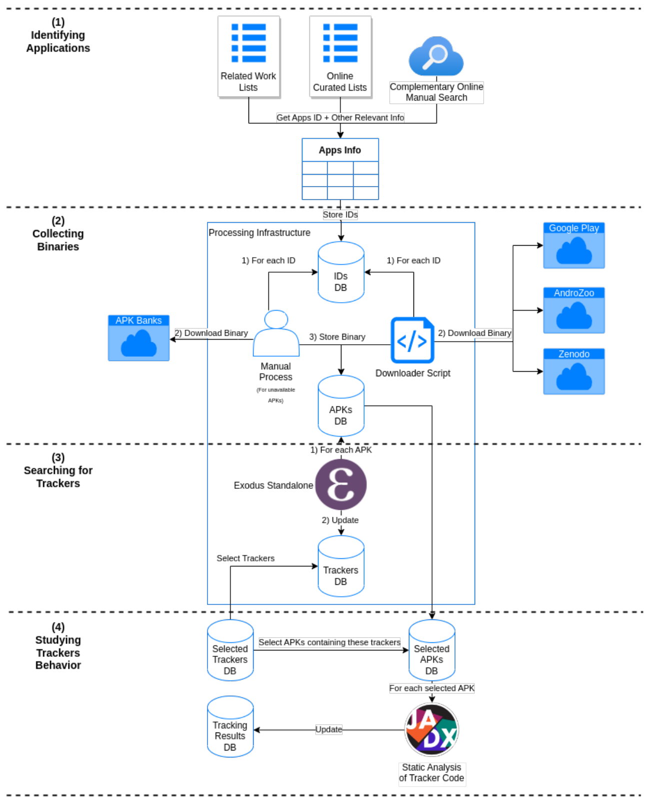


Figure 1. Tracker analysis methodology.

We adopt the term *tracker* in alignment with its usage in related literature, signifying a third-party library integrated into apps for analytics, advertisements, technical, or other functionalities, obviating the necessity for developers to code these functionalities themselves. These libraries collect information about the application, the device, and their usage, subsequently transmitting this data to their servers.

3.1.1 Identifying Applications

Our initial objective was to pinpoint COVID-19-related apps developed or sponsored by government entities, reputable international organizations, or recognized universities. Our aim was to encompass a broad spectrum, including applications from every relevant country, region, or independent jurisdiction, ensuring a comprehensive and impartial coverage. Our identification process commenced by referencing related works, including [AWO Agency, 2021; Cho *et al.*, 2020; Alfayez *et al.*, 2021], and consulting curated online lists such as [MIT Technology Review, 2020b,a]. This foundation was further enriched through our systematic online search, ensuring comprehensive coverage of applications across relevant countries and regions. We accorded special attention to the validation of the apps incorporated in our study. A meticulous manual verification process ensured that each selected app was genuinely associated with the context of the COVID-19 pandemic. This approach diverged from automated crawling methods utilizing keywords used in other studies, where, for instance, a search using the keyword “corona” yield results unrelated to COVID-19. In total, we identified 619 Android apps. Notably, according to Google Play download figures, 80 of these apps garnered downloads ranging from 1 million to 5 million times, 14 apps fell within the 5 million to 10 million download bracket, 21 apps recorded downloads between 10 million and 50 million, one app reached the 50-100 million range, and another app surpassed the 100 million download mark. This task was conducted between March 1st and March 15th, 2022.

3.1.2 Collecting Binaries

The collection, processing, and storage of binaries (APK files) were executed utilizing the infrastructure of *Cluster UY* [Nesmachnow and Iturriaga, 2019]. Our primary data source was *Google Play Store*. We aimed to maximize the number of binaries downloaded from this source to ensure file integrity. However, due to certain apps being unavailable for download in our region or having been removed from the store, we also explored three additional sources. *AndroZoo*, a collection of Android apps maintained by Université du Luxembourg [Allix *et al.*, 2016], was one of our secondary datasets. Additionally, we utilized a dataset created by Wang et al. [Wang *et al.*, 2021a] for their previous study on malware in COVID-19 apps, available on the open data repository *Zenodo* [CERN, 2013]. Some binaries were not accessible through the aforementioned sources, consequently, we opted to download them manually from APK repositories such as *Apksos*, *Apkpure*, and *Apkgk*. Out of the initially identified 619 apps, a total of 595 binaries were successfully collected between March 15th and March 30th, 2022. The breakdown includes 356 from *Google Play*, 152 from *AndroZoo*, 8 from *Zenodo*, and 79 that were manually downloaded. For a comprehensive list of collected binaries, please refer to the list we made public [Serrano *et al.*, 2023a].

3.1.3 Searching for Trackers

To test whether trackers were used within the collected binaries (RQ1), we employed the *exodus-standalone tool* [Exodus, 2022a]. This tool operates by taking an APK file as input and generating an output that includes a comprehensive list of trackers detected within the binary. The tool employs signature matching in its process to detect trackers within the binary. It extracts Java classes from the APK file and, for each identified class, compares the package name with a database of previously identified trackers [Exodus, 2022b]. By configuring the tool output in JSON format and implementing an automated process, we generated our dataset of trackers identified in the analyzed apps. In total, 58 trackers were detected. This work was done the first week of April 2022.

3.1.4 Studying Trackers Behavior

Since our objective was to surpass the scope of previous studies and delve into the precise nature of data collected by trackers in the COVID-19 apps ecosystem (RQ2), we conducted reverse engineering of their code. Facilitating these activities and static analyses, we utilized *JADX* [jadx, 2022]. This tool, which decompiles Dalvik bytecode and generates Java source code from Android APK files, provides a GUI similar to an IDE (also a console appliance is offered) helping view decompiled code with highlighted syntax and other common functionalities related to static analysis. Hence, *JADX* enabled us to identify the specific classes in the APK file responsible for data harvesting. By tracing the data flow within the observed tracker package, we could identify the internet connection methods employed to transmit the collected data to their respective servers. Among the 58 identified trackers, we scrutinized the behavior of 22 by examining the binaries of apps where they were integrated. These analyses were performed from April 2022 to mid-July 2022. The selection of these 22 trackers was conducted randomly, while ensuring representation from each category of services provided, including analytics, ads, push notifications, among others. Upon examining the selected trackers, we endeavored to identify patterns, beginning with specific cases and converging toward overarching observed practices. The detailed outcomes of our analysis are described in Section 4, and the potential impact on user privacy (RQ3) is discussed in Section 5.

3.2 Limitations and Research Ethics

Our research encountered certain limitations, primarily stemming from the absence of automated tools facilitating behavioral analysis on Android apps giving precise and trustworthy results. Consequently, the inspections were conducted manually, and the knowledge and expertise gained was used as the basis for the development of the SAPITO tool. The comprehensive study of all trackers flagged by *Exodus* exceeded the scope of our investigation, potentially resulting in the oversight of some trackers. The mere presence of a tracker in the application binary does not guarantee its invocation and execution. In our study, when a tracker was identified in the code, we operated under the assumption that it would run, leading to data collection. These assumption might generate false positives, although it is deemed unlikely.

During our manual analysis of trackers, we made diligent efforts to identify all data being collected. However, given the intricate nature of the task, it is conceivable that certain pieces of information may have been inadvertently overlooked. If a tracker aimed to obscure its code through reflection and dynamically downloading strings employed in class and method calls, this study would likely become unfeasible. Notably, we did not encounter this specific challenge in the trackers we analyzed.

On the other hand, our focus was specifically on Android apps, as there are currently no reverse engineering and decompiling tools for other platforms, such as iOS, that possess comparable capabilities.

Finally, our research adhered to stringent ethical procedures. To comprehend the trackers' behavior and discern the collected information, we conducted reverse engineering steps on the app binaries (APK files). The scope of these activities was confined to identifying the data collected from the user, their device, and their app usage, subsequently transmitted to the tracker servers. No other actions were taken with the binaries. We accessed all platforms used in our study, such as *Google Play* and *Exodus* website, in strict compliance with their respective terms of service. Lastly, our approach maintained neutrality towards trackers, apps, and countries. We aimed to cover every conceivable country, region, and jurisdiction, analyzing every available application within our capacity. We conducted manual investigations on as many trackers as possible.

4 Findings

In this section, we present the findings of our research. We delve into the specifics of the analyzed trackers and engage in a discussion regarding the privacy concerns arising from the use of these tools in the implementation of COVID-19-related applications.

4.1 Observed Methodology of Trackers

The analyzed trackers revealed consistent patterns in their data harvesting methodologies. This section details the most prevalent approaches, examining how these trackers collect data, when data collection happens, and the methods employed to transmit data to their servers. It is essential to note that the discussed perspectives are not mutually exclusive; therefore, a tracker may employ a combination of real-time, event-triggered, and centralized approaches in data collection.

How: Real-Time vs Store & Retrieve

- **Real-Time Collection:** Trackers employing this methodology collect information in real-time. They initiate the data collection process through a specific method in a class, creating a chain of method calls that culminates in the final method. In this last method, the gathered data is sent to their servers through a POST request.
- **Store and Retrieve Collection:** In this methodology, trackers initially collect information and promptly store it in classes' fields or internal databases. Subsequently, at a later point, the tracker queries the previously stored data and transmits it to its servers through POST requests.

When: Event-Based vs General

- **Event-Based Collection:** Trackers employing this approach harvest data when specific predetermined events occur. This data encompasses information relevant to the event as well as any other pieces of information that the tracker or app developer intends to collect.
- **General Collection:** Under this approach, trackers collect general information about the device, the user, the app, and its usage without waiting for specific events within the app.

Where: Centralized vs Iterative vs Decentralized

- **Centralized Collection:** In this scenario, all information is gathered from a singular point within the tracker, typically within a single method of a class. Leveraging the advantages of polymorphism in Object-Oriented Programming, certain trackers invoke the same method to collect data. However, the data collected varies based on the class where that method is implemented.
- **Iterative Collection:** Trackers employing an iterative approach gather data in a sequential manner, collecting information in multiple methods called in succession. With each successive call, the data harvested is added to the set of collected data from previous methods in the chain.
- **Decentralized Collection:** Under this approach, the tracker harvests data at distinct points in its code, which are unrelated to each other. For instance, in one method, it may collect and send device information, while in a different class and method, it may harvest and exfiltrate data about specific app events.

Exfiltration: Centralized Connections vs Decentralized Connections

- **Centralized Connections:** In this case, although the tracker could collect the information in a centralized or decentralized way, the requests that are sent to its servers to exfiltrate data leave from a single class.
- **Decentralized Connections:** Trackers that use this approach send the information using requests from methods that belong to more than one class. In general, the data harvesting happens in a decentralized way for cases in this category.

4.2 Applications and Trackers Statistics

We identified 58 different trackers within 402 of the 595 apps analyzed. Table 2 illustrates that trackers detected by *Exodus*, such as *Google Firebase Analytics* and *Google CrashLytics*, are prevalent in a substantial number of apps within the dataset, particularly the former (present in almost two out of three apps). Remarkably, nearly half of the identified trackers—28 out of 58—were included in no more than two apps.

We have also studied the correlation between the most included trackers. This can be appreciated in Fig. 2: the value of the cell (i, j) indicates the proportion of cases in which, if tracker i was present in an application, so was tracker j. It can be seen how using libraries of the same provider is a common practice, especially in the case of *Facebook*. Also, it is interesting to note what can be inferred from Table 2: the dependence on *Google Firebase Analytics* SDK, unless the libraries from *Microsoft* are used, or the specific case of *Bugsnag*, which shows no usage correlation with other trackers. The opposite happens to *Amplitude* since the chart indicates that when that tracker is included, SDKs from *Facebook* and *Google* are also used by the developers.

Trackers	**Applications**	**Percentage**
Google Firebase Analytics	357	60.00%
Google CrashLytics	180	30.25%
Google AdMob	47	7.90%
Facebook Login	40	6.72%
Facebook Share	38	6.39%
Facebook Analytics	34	5.71%
Google Analytics	28	4.71%
OneSignal | HMS Core	24	4.03%
Microsoft Visual Studio App Center Crashes	22	3.70%
Microsoft Visual Studio App Center Analytics	20	3.36%
Facebook Places	18	3.03%
Facebook Ads | Google Tag Manager | Amplitude	14	2.35%
OpenTelemetry	13	2.19%
AltBeacon | Bugsnag	12	2.02%
Matomo	9	1.51%
Segment	8	1.35%
Mapbox | Branch	7	1.18%
New Relic | Braze (*)	5	0.84%
MixPanel	4	0.67%
Splunk MINT | Pushwoosh | Facebook Flipper | Airship | Flurry	3	0.50%
Esri ArcGIS | Demdex | AppMetrica | Appcelerator Analytics | AppsFlyer | Adobe Experience Cloud | HyperTrack | Bugfender | County	2	0.34%
Startapp | AdColony | Radius Networks | RjFun | Heap | MOCA | Kontakt | ComScore | LotaData | Snowplow | Pusher | Appnext | Split | Instabug | Conversant | Scandit | TNK Factory | Bolts | Lotame |	1	0.17%

Table 2. Number of applications (out of 595) in which the identified trackers were found, together with their percentage of occurrence. (*) Braze was previously known as Appboy, which appears with this name in the binary packages. Same with ex Urban Airship now known as Airship.

Given their functionality and services, developers are encouraged to add SDKs to their apps. The trackers in our study provided the following services (based on their main functionality):

- **App Usage, Audience and Engagement:** analytics of how the app is used, and by who, events monitoring and tracking, user engagement (26 trackers);
- **Crash Reporting and Monitoring:** monitoring of resources, logs, crashes, and general app functioning (10 trackers);
- **Technical Functionality:** Specific functionality for the

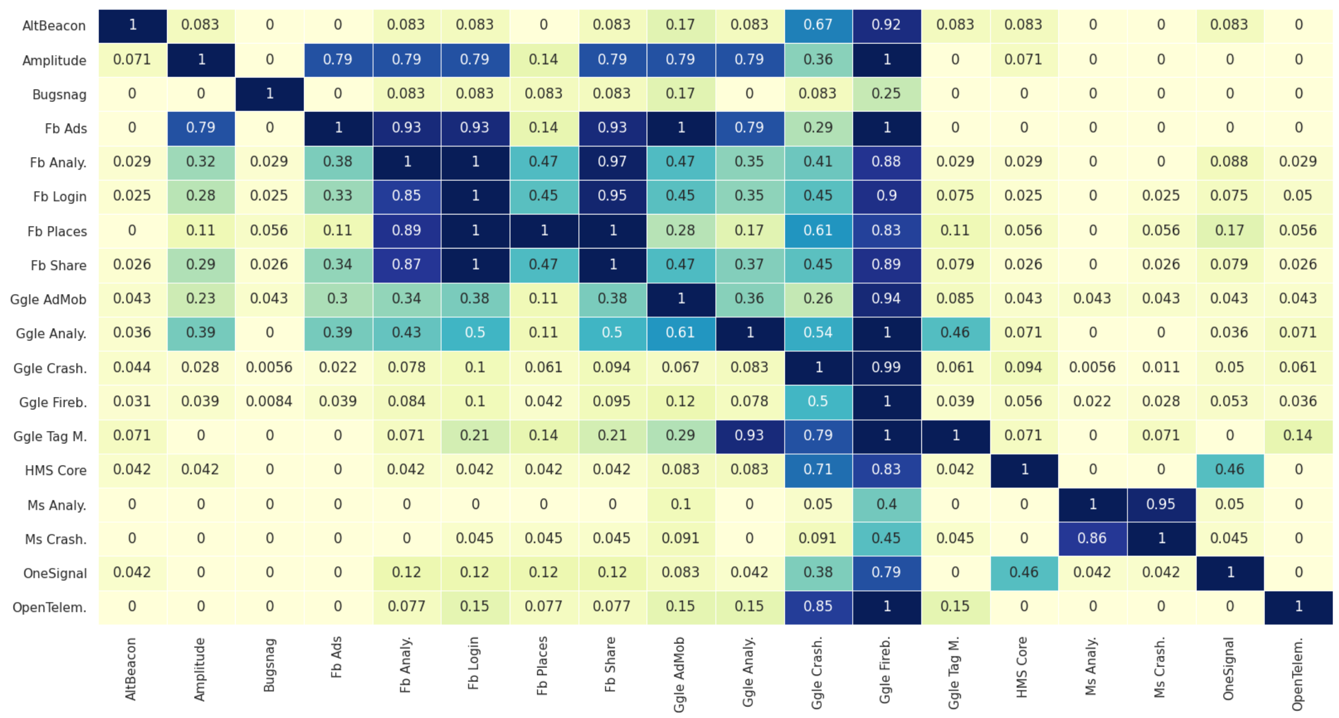


Figure 2. Proportion of concurrent usage of trackers.

user that is integrated into the app (e.g., maps, beacons, and code scanning) (8 trackers);

- **Mobile Advertising:** distribution of ads into the app, monetization (5 trackers);
- **Devel. and Back-end Framework:** Libraries for ease of app development and back-end services (e.g., hosting, storage, and distribution) (3 trackers);
- **Push Notification:** push notification and other messaging channels (3 trackers);
- **Social Media Integration:** integration with social media within the app (e.g., login and content sharing) (3 trackers).

Table 3 presents the frequency of inclusion of trackers in the various categories described above within the studied apps. It is evident that libraries aiding in the development and maintenance of apps in production were extensively employed. However, trackers providing advertisement services, whose relevance to the studied apps is debatable, were included 77 times.

Main Service	**Apps Count**
Devel. and Back-end Framework	382
Crash Reporting and Monitoring	242
App Usage, Audience, and Engagement	156
Social Media Integration	96
Mobile Advertising	77
Push Notification	28
Technical Functionality	27

Table 3. For each service category, a count of times trackers belonging to these categories were included.

From the viewpoint of the apps, as shown in Table 4, almost one-third of them do not include any tracker. Approximately half of the apps contain one tracker at most if any, and around 80% of the apps include no more than two trackers. Conversely, 30 apps had over five trackers. Table 5 illustrates the 15 apps that presented the most trackers in our dataset, ranging from 11 to 9 trackers. The purposes of these apps vary, with some having been downloaded hundreds of thousands or even millions of times.

Trackers Count	**Applications**	**Percentage**
None	193	32.44%
One	123	20.67%
Two	160	26.89%
Three	49	8.24%
Four	22	3.70%
Five	18	3.03%
More than five	30	5.04%

Table 4. Number of trackers found in the studied applications, and their respective percentage.

Amid the urgent deployment of digital health apps to curb the virus's spread at the onset of the pandemic, it would not be surprising if applications initially heavily relied on third-party libraries and subsequently removed or minimized their use in later releases. To investigate this, we leveraged historical information from application reports on *Exodus*. We queried the number of trackers detailed in each report and compared the counts between the first and the last available report to determine whether the tracker numbers per application decreased over time. Our aim was to assess whether, following the successful containment of the virus, governments, organizations, and other providers of COVID-19-related apps enhanced their privacy aspects. The results, outlined in Table 6 for apps with more than one privacy report (153 in total), reveal that approximately 16% (25 apps) decreased the count of included trackers, with 6 of these 25

Application	Region	Goal	Trcks.	Downls.
WebMD: Symptom Checker	International	Info.	11	10M+
COVID Symptom Study	UK	Studies	10	1M+
Korona Önlem	Turkey	Self-Diagn.	10	-
C Spire Health - UMMC Virtual COVID-19 Triage	USA	Info.	10	10K+
Sydney Care	USA	Info.	9	-
Manitoba Immunization Verifier	Canada	Cert. Verif.	9	50K+
Manitoba Immunization Card	Canada	Cert.	9	50K+
QMUNITY	Malaysia	Contact Tracing	9	-
Ministry of Health, Trinidad and Tobago	Trinidad and Tobago	Telehealth	9	1K+
CoVerified	USA	Cert.	9	10K+
SMART Health Card Verifier	USA	Cert. Verif.	9	100K+
SafeEntry (Business)	Singapore	Contact Tracing	9	100K+
Covid-19 Cuernavaca	Mexico	Info.	9	-
WHO LENA	International	Health-workers	9	1K+
RBC-C19	Rwanda	Cert.	9	10K+

Table 5. Top 15 applications with most trackers included. The count of downloads at the moment of writing this article is included, showing "-" if the application is no longer in Google Play.

having no trackers detected in their latest report. About 14% (21 apps) increased their tracker count, including 6 apps that initially had no trackers but later included at least one. The majority, around 70% (107 apps), maintained the same number of trackers between their first and last report, with 56 of these including trackers in their binaries.

Case	Status	Count
Decrease	Trackers at end	19
	No trackers at end	6
Increase	Trackers at start	15
	No trackers at start	6
Equal	Trackers	56
	No trackers	51
One report	-	184
No Report	-	258

Table 6. Increase and decrease of trackers per application, based on *Exodus* historical reports.

4.3 On Data Collection

Our key findings reveal that user, device, and usage data were systematically tracked. The operational procedure of trackers is illustrated in Figure 3. The tracker's company provides mobile developers with an SDK that imparts specific functionality upon integration into the application (1). Simultaneously, they configure the SDK server to establish connections when in use (2). Later, a developer—likely representing a government, international organization, or sponsored entity— incorporates the SDK into its COVID-19-related application (3), subsequently releasing it on Google Play Store (4). Users, motivated by governmental mandates, compelled by living circumstances, driven by fear of misinformation, or other incentives, download these COVID-19 apps (5) and start using them (6). While in operation, the SDK code executes, including the segment dedicated to information tracking, and eventually uploads the harvested information to the SDK servers (7). Finally, the company behind the SDK utilizes the collected data by storing it (8a), for example, to fingerprint the device [GetSocial, 2022], processing it for user profiling (8b) (*Engagement Data* in [Branch, 2022]), or simply sharing it with third parties for revenue (8c) (point 9 in [Startapp, 2022]).

To precisely ascertain the types of information harvested by the trackers, we meticulously examined their code in the apps listed in Table 7. Considering that SDKs included in the apps may differ across releases, and the code of the tracker itself may undergo changes over time, we also provide details regarding their respective versions.

Our findings categorize the data gathered by trackers into the following:

- **Android Advertisement ID (AAID):** In this category, we include the *Android Advertisement ID* and whether its tracking is limited.
- **User ID:** Identification artifacts in this category are primarily based on *UUID* variants. The AAID or user ID was sometimes directly used from the main application code. In some cases, an ID was assigned at installation time or was set with a push notification.

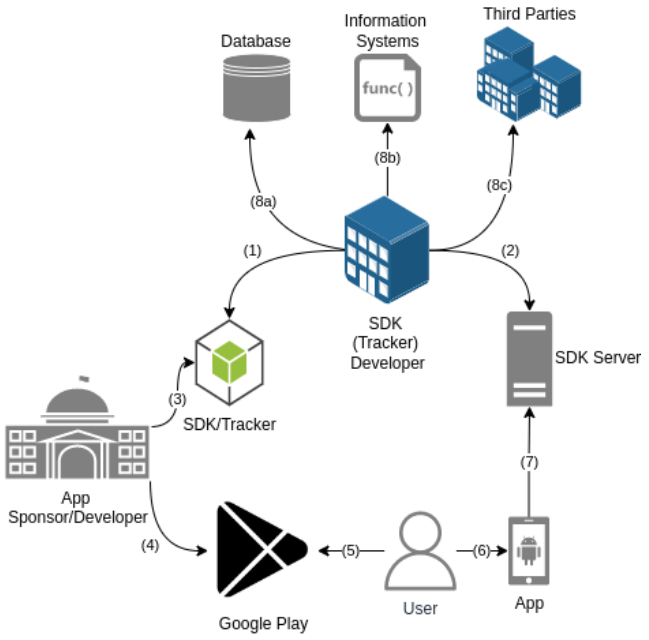


Figure 3. Tracker data collection example.

- **Location Software:** Tracking information approximating the user's location based on software configurations of the device. This includes calls to track the *locale*, *language*, and *timezone*, *country code*, *input languages* and *daylight saving*.
- **Location Hardware:** This category groups calls to get the user's and device's exact geographic location.
- **Device Software:** Information related to the operating system (its *name*, *version*, *architecture*, and *build*) and the *user-agent* used at connection time. In general, the *OS* name was hardcoded as *"Android"* in the trackers' code.
- **Device Hardware:** Information about the physical device, including the device *manufacturer*, *model*, and *brand*, *name*, *id*, *board*, *display* and if it is a *tablet or phone*. Additionally, tracking of the *SIM card* is included in this category.
- **Android Package Kit (APK):** Information about the application, including the *package name*, *version name*, *version code*, *application bundle*, *permissions granted*, *environment*, *development framework*, if it is an *instaApp* [Android, 2021], the *app store*, and its given ID by the tracker on its platform.
- **Applications/Processes:** Some trackers gathered information about the *activities running*, the *installed apps*, the *state* of the application, the *threads* running, and the *Java Virtual Machine*.
- **Disk/Memory:** There were some trackers that gathered information about the *disk*, the *filesystem* and the *memory* of the device.
- **Network:** Calls the trackers made to get information about the *carrier*, the *network* used by the device, the *wifi*, and *bluetooth*. Some trackers collect a large number of different elements related to this category (over one hundred).
- **Screen/Audio:** Information about the *display* and its *orientation* was commonly harvested. There was also information related to the audio, like the *ringer mode*, if an *earplug connection* exists, and the *volume*, among others.
- **Rooted/Jailbroken/Emulated/Simulated:** Some trackers attempted to identify if the device was *rooted/jailbroken* or if it was an *emulation/simulation*.
- **Time:** Time-related tracking activities, including the time of the *application install*, its *last update*, or other *events* that the trackers monitor. Examples include *timestamps* and *session duration*.
- **Battery:** Information about the battery was also tracked, namely its *level* and if it was being *charged*.
- **Software Development Kit (SDK):** Trackers collected data about themselves, such as their *version* and *flavor*, often hardcoded in their code.
- **Others:** Trackers harvest various dissimilar types of information from the device and its usage, including *metadata* and event *types* associated with the events they monitor.

Tracker	Tracker Version	Application Package	App. Version
AdColony	4.1.2	covid19.cuernavaca	9.8
Airship	5.1.0	au.com.vodafone.dreamlabapp	3.3.1.3218
AltBeacon	2.16.4	gov.georgia.novid20	1.0.467
Amplitude	2.23.2	sg.gov.tech.safeentry	0.11.0
AppNext	2.4.5.472	covid19.cuernavaca	9.8
Branch	3.2.0	ca.bc.gov.health.hlbc.COVID19	1.42.0
Braze	8.0.0	com.clearme.clearapp	1.11.1
Bugsnag	5.1.0	gov.adph.exposure-notifications	1.10.0
Flurry	11.5.0	mu.mt.healthapp	2.0.103
Google AdMob	12.4.51	tr.gov.saglik.korona-onlem	1.0.3
Google Firebase Analytics	17.0.0	ca.ontario.verify	1.1.1
Google Tag Manager	5.06	az.gov.my	1.6.1
Mapbox	9.6.2	com.healthcarekw.app	2.1.9
Matomo	N/A	cy.gov.dmrid.covtracer	3.3.12
MixPanel	4.8.7	com.moh.alert.ramzor	1.15.0
New Relic	6.3.1	ar.gob.coronavirus	3.5.32
OneSignal	3.12.3	uy.gub.salud.plancovid-19uy	9.1.1
Open Telemetry	0.21.0	es.gob.asistenciacovid19	1.0.11
	0.21.0	au.gov.health.covid19	1.4.10
Pushwoosh	6.3.6	gov.cdc.general	3.1.4
Segment	4.8.2	com.thecommons-project.smarthealthcard-verifier	1.0.27
Splunk MINT	5.0.0	et.gov.moh.oppia.covid	7.3.0-et.2.int
Startapp	4.5.0	covid19.cuernavaca	9.8

Table 7. Applications that were analyzed to detect tracking behavior. We studied two applications for *OpenTelemetry* to validate our findings. In the case of *Segment*, we found a pattern in applications developed using the *EXPO* framework, where several wrappers for trackers (including *Segment*) were automatically included in the applications.

In Table 8, **we provide a detailed account of the information harvested by each tracker through our analysis of their code**. The tracked data is categorized based on the previously outlined classifications. *AltBeacon*, *Google Tag Manager*, and *OpenTelemetry* are highlighted in green as we could not identify tracking behavior in their code. Conversely, *Google AdMob* and *Startapp* are marked in a red hue, signifying their comprehensive tracking of information across all described categories.

4.4 On Push Notifications

A noteworthy concern we identified pertains to the utilization of push notifications for conveying sensitive information to application users. Specifically, considering the nature of the applications under scrutiny, push notifications have the potential to encompass confidential details such as exposure alerts, COVID-19 test results, and health-related notifications. Analogous to the data collection functionality, we illustrate this behavior in Figure 4. Initially, the SDK developer provides its push notification SDK (1) and associated server (2). Subsequently, the developer incorporates the push notification SDK into the application (3) and releases it on the store (4). Users proceed to download (5) and use the application (6). When dispatching a new push notification to all users, a specific group, or a targeted user of the application, developers must designate the recipients and specify the message content using the service platform on the push notification server (7). This implies that the message is transmitted in plaintext to the SDK provider, who can potentially store, process, or transmit this information to a third party (8).

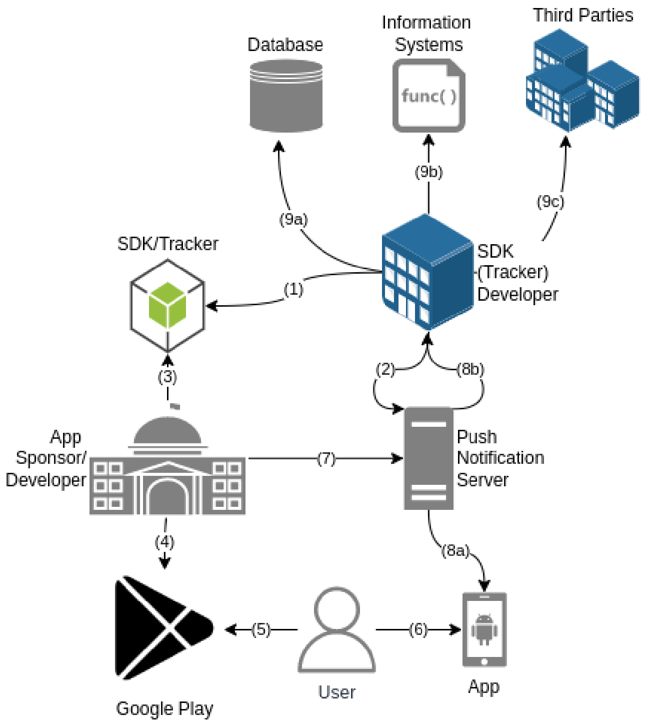


Figure 4. Tracker push notification example

Among the 22 trackers analyzed, six of them—namely, *Airship*, *Braze*, *Google Firebase Analytics*, *Flurry*, *OneSignal*, and *Pushwoosh*—integrated push notification services into the apps. Interestingly, our examination of the services' documentation did not reveal any explicit mention of end-to-end encryption, except for the case of *Pushwoosh* [Pushwoosh, 2019]. This matter will be detailed further in Section 5. Additionally, other trackers identified in our study that offer push notification services include: *Adobe Experience Cloud*, *AppMetrica*, *Countly*, *Facebook Analytics*, *HMS Core*, *LotaData*, *MOCA*, and *Pusher*.

Tracker	**AAID**	**User ID**	**Loc. SW**	**Loc. HW**	**Dev. SW**	**Dev. HW**	**APK**	**A/P.**	**D/M**	**Net.**	**S/A**	**R/E**	**Time**	**Batt.**	**SDK**	**Oth.**
AdColony	✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	✓
Airship		✓	✓	✓	✓	✓	✓	✓		✓	✓		✓		✓	✓
AltBeacon																
Amplitude	✓	✓	✓	✓	✓	✓	✓			✓			✓			✓
AppNext		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓
Branch	✓	✓	✓		✓	✓	✓			✓	✓		✓		✓	✓
Braze	✓	✓	✓	✓	✓	✓	✓			✓	✓		✓		✓	✓
Bugsnag		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flurry	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓
Google AdMob	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Google Firebase Analytics		✓			✓	✓	✓								✓	✓
Google Tag Manager																
Mapbox		✓		✓	✓	✓		✓		✓	✓		✓	✓	✓	✓
Matomo		✓	✓			✓	✓				✓		✓		✓	✓
MixPanel					✓	✓	✓			✓	✓				✓	
New Relic		✓	✓		✓	✓	✓	✓	✓	✓	✓				✓	✓
OneSignal	✓	✓	✓	✓	✓	✓	✓			✓		✓				✓
OpenTelemetry																
Pushwoosh		✓	✓		✓	✓	✓			✓		✓	✓		✓	✓
Segment	✓	✓	✓		✓	✓	✓			✓	✓				✓	✓
Splunk MINT		✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Startapp	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 8. Type of information harvested by the trackers and sent to their servers.

5 Discussion

In this section, we elaborate on our findings and evaluate their potential impact from the users' standpoint. In addition, the issue of informed user consent is analyzed in light of both apps and trackers privacy policies.

5.1 Impact of Data Collection

Device fingerprinting, coupled with user segmentation and attribution, is a well-documented practice. Academic research in this domain has proposed effective fingerprinting mechanisms, aiming to enhance authentication [Wu *et al.*, 2016]. Additionally, trackers openly acknowledge such practices in their privacy policies, explicitly stating the purpose of data collection as "device identification and attribution" [Branch, 2022]. This becomes particularly streamlined when they collect the Android Advertising ID (AAID) of the device.

However, in the context of COVID-19-related apps, this tracking and identification behavior poses an exceptionally high risk to users and the confidentiality of their health-related sensitive data. Notably, 34 of the apps we analyzed were primarily utilized for quarantine enforcement, and among them, 29 harbored trackers that, in addition to other data points, collected the APK identifier. Consequently, if the entities behind these trackers were aware that these APKs were associated with quarantine enforcement, they could deduce that the device user was infected with COVID-19. This implies that, unwittingly, governments shared their citizens' COVID-19 infection status with third parties through these apps. Furthermore, considering that certain trackers collect location information, there is a potential for them to have precise GPS-level details about where users were infected with the virus. This line of reasoning applies to apps focused on vaccination scheduling and COVID-19 certification as well. These trackers could unveil information about users' visited locations, potentially identifying instances when users frequented hospitals. Additionally, since some trackers capture the SSID of Wi-Fi hotspots within range, they could corroborate the user's presence at specific locations.

5.2 Impact of Using Push Notification Services

As discussed in Section 4, developers are required to transmit the message content of push notifications to service providers in plaintext. Consequently, messages containing sensitive health information such as *"Your test results were positive..."* or *"You are scheduled for your X COVID-19 vaccination dose..."* hold the potential to inadvertently expose users' confidential health details to third parties. What exacerbates the issue is that application users are generally unaware of and have not consented to the specific disclosure of this information to external entities.

Even in instances where encryption was employed to safeguard the confidentiality of notification contents, the methodology raises potential concerns. Utilizing asymmetric cryptography, a key pair can be generated on the user's device, with the public key sent to the SDK servers. Consequently, any push notification becomes end-to-end encrypted, ensuring that only the respective device can decrypt the message and display it to the user. However, the critical aspect lies in defining the endpoints of this end-to-end encryption scheme. This approach would effectively preserve privacy if one end were the device and the other the application developer. Unfortunately, in all examined cases where this was implemented, encryption occurs at the service provider servers. As a result, they process the notification message in plaintext before encrypting it, potentially compromising user privacy during this intermediate stage.

Developers facing the challenge of transmitting sensitive COVID-19 information through push notifications should reassess their requirements. Are push notifications absolutely necessary? Considering alternatives such as email or SMS might encounter similar challenges, but opting for a phone call could circumvent these issues. Another approach is to keep push notification content as a simple alert, prompting users to open the application for detailed information. Additionally, developers could explore the use of genuine end-to-end encryption libraries, such as *Capillary* [Perera and Hogben, 2018], ensuring secure communication from the developer servers to the user's device.

5.3 Data Privacy Posture from Governments

As discussed in Section 4, while data privacy holds significant importance (acknowledged as a fundamental human right [European Data Protection Supervisor, 2022]), in the context of a global pandemic posing a severe threat, there might be a shift in priorities for governments and individuals. It is understandable that, in the urgency to launch an app early to aid pandemic control, certain trackers could be justified within the app's code.

However, what raises concerns about the due diligence of data privacy from governments and reputable organizations is the lack of reduction in the number of trackers included in these apps after their initial release, especially as the pandemic comes under control. Additionally, Table 6 reveals instances where the number of trackers in some apps increased over time during the pandemic.

Moreover, we conducted a preliminary examination of several application privacy policies, revealing instances where the presence of trackers within the application was not mentioned in the apps privacy policies [MoH Turkey, 2020; MCIT Indonesia, 2020; MoI Qatar, 2020]. In contrast, some developers explicitly disclosed the inclusion of third-party libraries, enabling users to be aware of their presence in the apps. For instance, one app stated, *"Smart app uses third-party services that declare their Terms and Conditions. Link to Terms and Conditions of third-party service providers used by the Covid-19 DXB app: Google Play Services, Google Analytics for Firebase, Firebase Crashlytics"*[Dubai Health Authority, 2020]. Similarly, another app communicated, *"The app may transfer data relating to the use of the app and the device to Firebase (to detect problems in the app) with the user's consent. The transfer is subject to the following privacy policy: https://firebase.google.com/support/privacy. No data related to the content of the certificates will be transmitted."* [CovidSafeBE, 2020]. A more comprehensive analysis and comparison of privacy policies could offer valuable insights that help improve the transparency and clarity of privacy policies.

Moreover, several governments conducted Data Protection Impact Assessments (DPIA) [Bock *et al.*, 2020; HSCNI Northern Ireland, 2020b; HSE Ireland, 2020] to identify and minimize data protection risks associated with their COVID-19-related apps. This represents a significant stride toward the international standardization of efforts to protect data privacy and implement privacy-by-design methodologies. Regarding the analysis of tracker inclusion within the application, some reports explicitly state that *"...there will be no third-party trackers gathering personal data in the app..."* [NHS United Kingdom, 2020]. Others evaluate specific elements, such as the impact of using *Firebase* for push notifications or the SDK of *Microsoft* to diagnose performance and stability issues [MoH New Zealand, 2020]. However, similar to privacy policies, certain DPIAs omit the analysis of the impact of trackers [HSCNI Northern Ireland, 2020a; Koronavirus app Croatia, 2020].

It can be inferred that certain users were not fully aware of the privacy policies of the apps they were compelled to install, and that some of the apps were developed without fully analysing the privacy implications of including trackers.

5.4 What do the Trackers Declare?

We discovered that, in general, trackers publicly disclose the data they collect, explicitly enumerating the collected items in their privacy policies. However, some trackers claim to collect fewer items than we found during our code analyses. The overall outcome of this study is presented in Table 9. The list of analyzed policies can be accessed on our public project [Serrano *et al.*, 2023b]. We posit that Google's requirement for their *Google Play's new safety section* has motivated (or, to a certain extent, mandated) trackers to transparently indicate the data they collect with their SDKs.

Despite our best efforts, we were unable to find a disclosure by *Amplitude* and *Segment (Twilio)* regarding the data they collect from their Android SDKs. No privacy policy or data collection information was found for *AltBeacon* and *Open Telemetry* (although we also did not observe any tracking behavior from them).

Notably, some trackers, such as *Startapp*, openly declare that they **will share/sell** the tracked information from the users of the apps where their SDK is included: *"We will share, license, sell, transfer or make available your data (or part of it) with Advertisers, advertising networks, Business Partners and/or our Data Partners which may use it while serving you targeted and/or personalized advertisements..."*. It could be debatable that government-sponsored apps that are used to contain the spread of a global pandemic include these trackers.

5.5 Research Questions Discussion

As outlined in Section 1, our study aimed to address three primary questions concerning COVID-19 applications. We proceed to analyze each of them:

(RQ1) Are trackers being used in the COVID-19 app ecosystem? How did this usage evolve over time? As detailed in Section 4, trackers were extensively employed in COVID-19 mobile apps. We identified a total of 58 different trackers across 402 out of 595 analyzed apps, with some apps incorporating more than 5 trackers. It is noteworthy that

Tracker	Declares	Coincides
AdColony	Policy	No
Airship	Enumeration	Yes
AltBeacon	-	-
Amplitude	Policy	No Data
AppNext	Policy	No
Branch	Mix	Yes
Braze	Enumeration	No
Bugsnag	Enumeration	Yes
Flurry	Enumeration	Yes
Google AdMob	Enumeration	No
Google Firebase	Enumeration	Yes
Google Tag Manager	Enumeration	Yes
Mapbox	Policy	Yes
Matomo	Enumeration	Yes
MixPanel	Enumeration	Yes
New Relic	Enumeration	No
OneSignal	Enumeration	Yes
Open Telemetry	-	-
Pushwoosh	Enumeration	No
Segment	Policy	No Data
Splunk MINT	Enumeration	Yes
Startapp	Policy	No

Table 9. Trackers declaration of data collected with their Android SDK. Column *Declares* has the values *Enumeration* if they enumerate each element they track, *Policy* if they provide a general idea in their Privacy Policy, but without explicitly detailing the collected items, or *Mix* if they enumerate within the Privacy Policy. Column *Coincides* indicates if what they declare coincides with our findings.

almost one-third of the apps in our dataset did not include any trackers. Additionally, our historical research on *Exodus* reports revealed no evidence that the number of trackers included in the apps decreased over the course of the pandemic.

(RQ2) What information is tracked and how? Our in-depth analysis of the trackers' code allowed us to precisely identify the information being collected and transmitted to their servers. In the context of COVID-19-related apps, the information shared with third parties encompassed crucial details such as the *Android Advertisement ID* of the device, its specific *location* and *locale* configuration, operating system specifics (*model*, *brand*, *manufacturer*), and the device's *rooted* condition. Additionally, the trackers accessed and transmitted data related to the status of *memory*, *disk*, *screen*, *audio*, and *battery*, along with pertinent *network* details. Furthermore, the trackers extracted information about the *application* where they were integrated, including other *running apps and processes*.

Upon comparing these elements with the information disclosed by trackers in their data collection documentation, we observed instances where not all collected items were transparently disclosed. Moreover, the utilization of push notification services introduced the potential for the inadvertent disclosure of sensitive health-related information to the service providers, contingent on the content of the messages.

We also delved into the methodologies employed by the trackers, revealing that they adopt either real-time data harvesting or store and retrieve information at a subsequent point. The initiation of data collection may or may not be event-triggered. Moreover, trackers exhibit a spectrum of data collection approaches, spanning from centralized to decentralized methods. These centralized/decentralized methods are mirrored in how trackers establish connections with their back-end servers.

(RQ3) What are the potential impacts on users if trackers are in use and harvest that information? Section 5 details the potential ramifications of incorporating trackers and push notification services into COVID-19 apps. Our analysis leads us to the conclusion that, under specific conditions, there is a risk of disclosing sensitive health-related information to third parties, potentially contravening relevant regulations. For instance, by collecting the *app ID*, trackers could infer a user's COVID-19 infection status if the app was employed for quarantine enforcement or disease tracking. Furthermore, the collection of *location* information raises concerns about the identification of infected users' whereabouts, potentially revealing their healthcare provider. The use of push notification services also introduces another avenue for disclosing users' positive infection status to third parties.

Compounding these concerns, certain trackers openly admit to transferring or selling collected data to external entities, amplifying the impact on users. Notably, the inclusion of trackers in apps with millions of downloads magnifies the scale of affected users, reaching into the hundreds of millions.

6 SAPITO: a tool for information leaking analysis of mobile applications

As discussed earlier, the contemporary development of Android apps invariably involves the integration of third-party libraries, offering pre-built functionalities that expedite the development process. However, a potential problem emerges when some of these libraries turn out to be trackers.

From the standpoint of privacy and cybersecurity, identifying suspicious classes and methods responsible for data harvesting poses a formidable challenge in the examination of data exfiltration within third-party libraries. The task of conducting static analyses on binaries to identify risky behaviors within these libraries is time-intensive, given the obfuscated nature of code and the multitude of classes, methods, and fields present in binaries. This intricacy was encountered in our research.

Several tools are available to aid in identifying trackers and data leaks, each with its set of limitations. For instance, *Exodus* is a valuable tool that promptly identifies trackers within Android apps based on a pre-populated database containing known tracker signatures. While analyzing an app, if a package signature matches one in the *Exodus* database, that package is flagged as a tracker. However, this approach restricts the discovery of new trackers in the wild that have not yet been included in the database. Additionally, *Exodus* does not provide explicit details about the data that trackers harvest and leak.

In response to these challenges, we developed SAPITO (SDK Audit and Privacy Investigation TOol), a prototype designed to flag packages within Android apps that appear suspicious of leaking sensitive information related to the phone, the app, the user, and the app usage. Offered as an open-source tool, SAPITO is crafted to be user-friendly, particularly catering to privacy teams for auditing third-party libraries in Android apps to uncover potential data leakages. The tool emphasizes usability, ensuring that even non-technical analysts can leverage its capabilities.

It can be accessed at [Serrano, 2023].

6.1 Tool Description

6.1.1 Technology Behind SAPITO

The detection of dangerous SDKs is automated through a static analysis of the app's binaries, enabled by the use of *Androguard* [Desnos, 2022]. This tool facilitates the instrumentation of Android binaries analysis, providing a Python API with several methods that can be called to access the code and other information of the APK file. Further information is scraped from *Exodus* and *Google Play Store* websites in real-time. SAPITO is implemented in Python 3, and its detection rules are loaded as *JSON* files for easy updates. Moreover, it uses *Flask* for its web interface. Figure 5 displays the architecture of the tool.

Users can interact with the tool through its web interface or load it as a separate module in Python 3 to automate privacy and leakage checks.

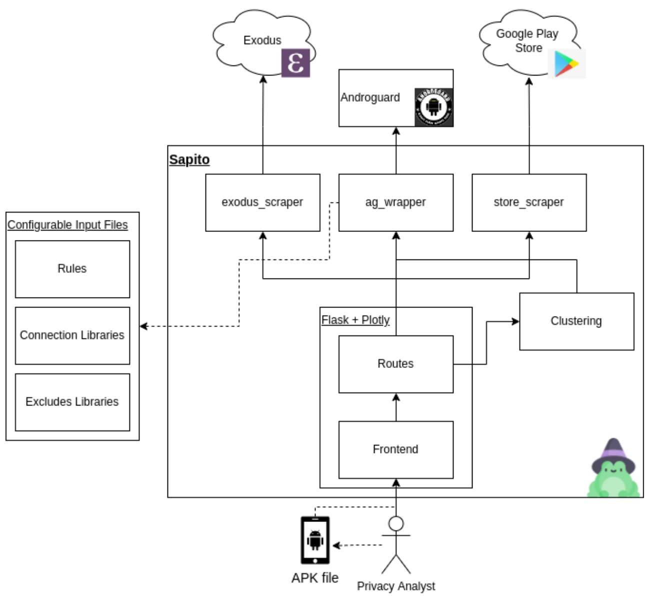


Figure 5. SAPITO's architecture

Ruleset Loading SAPITO looks for suspicious activity in third party-libraries based on instructions found in their code. These dangerous instructions that SAPITO must find in the code must be specified in a file called rules.json. By default, we provide a set of rules we found to be used by trackers to harvest data. The rules are grouped by the type of data harvested, for example:

- *location: ["Latitude", "Longitude", "Altitude"]*
- *Date: ["Date", "Time"]*
- *telephony: ["NetworkOperator", "NetworkType"]*
- *os: ["Version.Release", "Model", "Brand"]*
- *net: ["NetworkId", "WifiSSID", "TypeName"]*
- *Locale: ["Language", "Country", "Variant"]*
- *Thread: ["CurrentThread", "Id", "Name"]*
- *util: ["DensityDpi", "WidthPixels", "HeightPixels"]*
- *rooted: ["Rooted", "Jailbroken", "Simulated"]*
- *AdvertisingIdClient: ["AdvertisingIdInfo", "ID"]*

This file can be expanded or pruned based on the research needs.

6.1.2 What is SAPITO Capable of?

The primary objective of SAPITO is to automatically identify potentially suspicious third-party libraries in Android apps. This enables users to conduct detailed analyses of the activities undertaken by each third-party library included in the app in a swift, intuitive, and efficient manner. For instance, a developer might be interested in understanding the background activities of a library included in their app, while a privacy analyst would seek to comprehend the specific sensitive data being harvested. To accomplish this mission, SAPITO incorporates various analyses, as detailed below.

APK Loading This serves as the initial screen where users can load an APK file stored on their device into the tool. Once the binary is selected, SAPITO initiates its static reverse engineering analysis using the *Androguard* wrapper.

Main Report This page serves as the landing point after SAPITO completes its initial processing of the app. The tool automatically flags suspicious libraries, highlighting them based on the presence of classes that establish internet connections, make calls to obtain sensitive information, check for permissions or root status, utilize reflection calls, or handle notifications. In this report, the package names are color-coded, progressing from yellow to red based on the identified risky behavior. These third-party libraries are potential candidates for further examination, and users can select their checkboxes in the left panel to access more detailed information about them.

Packages Clustering SAPITO leverages unsupervised machine learning algorithms for package clustering. This analysis groups packages exhibiting similar cross-references, aiding in directing attention to specific libraries that may behave as trackers in the wild (for example, if these end up inside a cluster with known trackers). The model utilizes Principal Component Analysis (PCA) for feature reduction and KMeans for clustering. Technically, the model generates a dataset containing all cross-references present in the code of third-party libraries. Subsequently, PCA is employed to condense the dataset into two dimensions. The model then undergoes four runs of KMeans, iterating on the cluster number from two to five. The run with the optimal performance is selected. The model's output is visualized using the Python library *Plotly*, chosen for its user-friendly interface. To enhance visualization, markers for libraries already flagged as suspicious by SAPITO are differentiated, indicating that adjacent markers may also involve data harvesting code.

App Information General information is extracted from the app's manifest, providing insights into the permissions requested by the app. This is crucial because libraries integrated into the app might utilize these user-approved permissions without the user's awareness. For instance, while an app may legitimately require access to phone contacts, any function within the app, including code within third-party libraries, could read contact details after user approval. Key information in this section includes: *App Name*, *Main Activity*, *Activities*, *Services*, *Receivers*, *Providers*, *Libraries*, *Declared Permissions*, *Permissions*, *Requested AOSP Permissions Details*, *Requested Not AOSP Permissions Details*, *Uses Implied Permissions*, among others.

Google Play Information This option provides real-time details extracted from the *Google Play Store* page of the app. Similar to the previous analysis, this page offers additional information for the analyst. Details include: the app *URL in Google Play*, *Description*, *Summary*, *Installs*, *Real Installs*, *Score*, *Ratings*, *Reviews*, *Developer*, *Developer Email*, *Developer Website*, *Privacy Policy*, *Genre*, *Released*, *Version*, and selected users' *Comments* from the store platform.

Exodus Information This option presents live information from the *Exodus* website. Particularly valuable if prior analyses were performed on the *Exodus* platform, as it displays already-detected trackers for a given app version. It enables the study of trackers' evolution across different versions. Links to full reports, trackers' information, and trackers' webpages are included.

Library Cross-References SAPITO allows verification of cross-references made by each library. This feature aims to ensure that a library does not call dangerous native methods or fields to harvest sensitive data without justification. SAPITO highlights potential dangerous calls in red and suspicious cross-references in yellow for a more user-friendly experience. Dangerous calls are cross-references to functions/fields used by trackers to harvest sensitive data, while suspicious calls are cross-references to classes where dangerous calls are included, even if the specific method/field is not called. This categorization is based on the knowledge we obtained in our research.

Library Permissions Check This analysis in SAPITO evaluates the permissions checks conducted by the selected library. It ensures that the library is utilizing only the necessary permissions for its intended functionality. For instance, a push-notification library should not require location access permissions, and SAPITO helps identify potential malicious behavior by pinpointing such instances where permissions may be exploited to harvest sensitive location data without proper justification [Android, 2023].

Library Rooted Checks SAPITO facilitates an investigation into whether the library performs checks to detect rooted or jailbroken devices, or if the device is being emulated or simulated.

Library Reflection Use SAPITO identifies instances where libraries make reflection calls, providing information on the specific method called and its parameters. Given that reflection can be misused to access sensitive information, this analysis is valuable for privacy and cybersecurity analysts. An example of such misuse was documented in [Wang *et al.*, 2021b].

Library Connections This feature of SAPITO examines network connection calls made by third-party libraries. The tool incorporates information about commonly used connection libraries, flagging any matches found in the code. This is crucial as third-party libraries making internet connections can potentially exfiltrate data.

Library Push Notifications Use SAPITO's final feature (as of now) delves into the notification service calls identified in the app's code, specifically within the selected packages. Notification services can pose a risk, as the content of notifications may be processed in plaintext by SDK servers, without the knowledge and consent of app users.

6.2 Use Case: Analysing Coronavirus UY app

6.2.1 App reference

Coronavirus UY [AGESIC Uruguay, 2020] stands as the official app provided by the Uruguayan government to combat the spread of the COVID-19 virus. It has been downloaded over one million times at the moment of this writing, being widely spread across the national population.

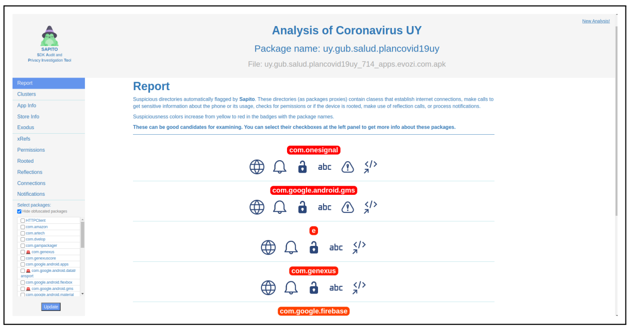
This app boasts a range of essential features, including contact tracing, exposure alerts, national COVID-19 statistics, vaccine scheduling, certificate storage, and telemedicine services, among others.

We highlight *Coronavirus UY* as a pertinent case for SAPITO due to its critical significance and widespread adoption in Uruguayan society. The utilization of SAPITO in such scenarios becomes pivotal, offering the ability to swiftly detect privacy risks, providing insights before deploying the app or its subsequent versions into production.

6.2.2 Analysis

Once the APK of the *Coronavirus UY* app is loaded into SAPITO and the binaries are analyzed, users are directed to the main report page. At this juncture, they can opt for various in-depth analyses of the app and its incorporated third-party libraries. Additionally, in the left-bottom panel, SAPITO enumerates the detected third-party libraries. Libraries with a red alarm preceding their name have been flagged by SAPITO as potentially dangerous packages, indicating a potential for tracking behavior. For this case, depicted in Fig. 6, the tracker *OneSignal* is visibly flagged, among other potentially hazardous third-party libraries. The report on this tracker reveals its engagement in internet connections (globe icon), notification processing (bell icon), permission checks (lock icon), use of reflection calls (abc icon), verification for rooted or emulated devices (danger sign icon), and invocation of sensitive classes in its code (snippet icon). With six categories of risky behavior present in this tracker, its name label is prominently displayed in solid red.

Selecting the clustering report on the left, it can be noted the formation of a cluster on the left side of the chart, with several other third-party libraries dispersed in the middle and on the opposite side of the graph, as illustrated in Fig. 7. Since the chart is interactive, hovering over any point with the mouse reveals detailed information. In this instance, *OneSignal's* point was highlighted, and given its status as a known tracker, the three adjacent points may also be potential trackers (one of them was Firebase, which was also analysed by the team, and proved to be a tracker).



Analysis of Coronavirus UY

Package name: uy.gub.salud.plancovid19uy

File: uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk

Report

Suspicious directories automatically flagged by **Sapito**. These directories (as packages proxies) contain clasess that establish internet connections, make calls to get sensitive information about the phone or its usage, checks for pemissions or if the device is rooted, make use of reflection calls, or process notifications.

Suspiciousness colors increase from yellow to red in the badges with the package names.

These can be good candidates for examining. You can select their checkboxes at the left panel to get more info about these packages.

com.onesignal

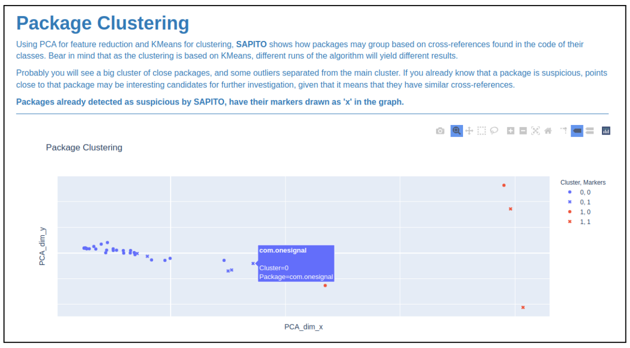
com.google.android.gms

e

com.genexus

com.google.firebase

Figure 6. Report page in SAPITO, where potential trackers are highlighted.



Package Clustering

Using PCA for feature reduction and KMeans for clustering, **SAPITO** shows how packages may group based on cross-references found in the code of their classes. Bear in mind that as the clustering is based on KMeans, different runs of the algorithm will yield different results.

Probably you will see a big cluster of close packages, and some outliers separated from the main cluster. If you already know that a package is suspicious, points close to that package may be interesting candidates for further investigation, given that it means that they have similar cross-references.

Packages already detected as suspicious by SAPITO, have their markers drawn as 'x' in the graph.

Figure 7. Cluster page in SAPITO, where trackers are grouped to detect similarities between them (cropped image of SAPITO's screen).

On the other hand, selecting the *App Info* option in the left panel, we found that the permissions requested by the app included:

- *android.permission.RECORD_AUDIO*
- *android.permission.ACCESS_NETWORK_STATE*
- *android.permission.WAKE_LOCK*
- *android.permission.RECEIVE_BOOT_COMPLETED*
- *android.permission.BLUETOOTH*
- *android.permission.FOREGROUND_SERVICE*
- *android.permission.CHANGE_NETWORK_STATE*
- *android.permission.INTERNET*
- *android.permission.VIBRATE*
- *android.permission.CAMERA*

Hence, dangerous scenarios may appear to the user in case a third-party library within the app (like *OneSignal*) would want to piggy-back on these permissions to perform actions without the user's consent (for example, recording audio, using the camera and connecting to the internet).

Following with the analysis, Exodus information analysis displayed four total reports. There could be appreciated that the last report contained four trackers highlighted by Exodus: *Google CrashLytics*, *Google Firebase Analytics*, *Huawei Mobile Services (HMS) Core*, and *OneSignal*. Furthermore, a comparison of the evolution of trackers' usage over the app versions can be performed with this information.

Analysing the cross-references of *OneSignal*, in Fig. 8 a few of them can be appreciated. There can be seen a red (dangerous) call, getting the extras of an intent to, for example, get sensitive information, and a yellow (suspicious) call.

Observing the permissions requested by *OneSignal*, we found that in five times it checks or uses permissions. Of these five, the three permissions used are:

- *RECEIVE_BOOT_COMPLETED*
- *ACCESS_FINE_LOCATION*,
- *ACCESS_COARSE_LOCATION*.

In this case, the accessing to the device's location may be considered a privacy problem.

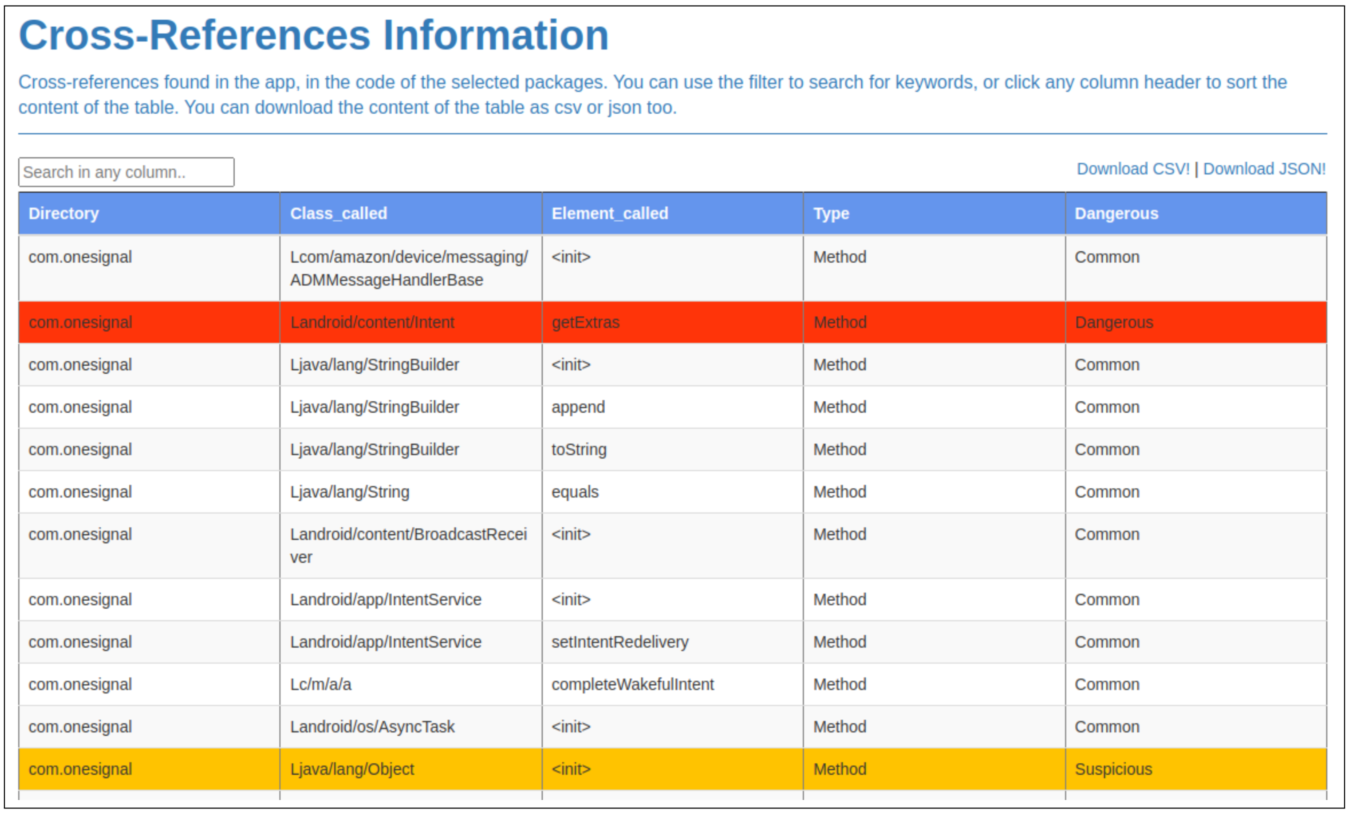
In the rooted checks analysis performed by SAPITO, it was found that *OneSignal* tests if it can run the *su* command, and if it has access to several restricted directories (meaning that it has root-level permissions), to identify whether the devices are rooted. At the same time, it can be seen how these calls originate in a method named *a*, from the class *h2* of package *com.onesignal*, showing the obfuscation done by the tracker to make the reading of its code as difficult as possible.

The use of reflection made by *OneSignal* was highlighted by SAPITO as well. Among other cases, it was detected how at method *<init>* (class constructor) from class *i2*, it calls to method *d* of class *com.amazon.device.iap.internal.d*. Obfuscation of code undermines the understanding of these calls but, initially, these may be suspicious.

Regarding internet connections, SAPITO enumerated every place where *OneSignal* called to the *java/net* connection library, specifically to the class *HttpURLConnection* and its methods for opening connections, sending data, getting responses and closing the connections.

Finally, in the push notification analysis, SAPITO described how *OneSignal* processes push notifications within its code. Therefore, and given the nature of this app (coronavirus control and e-health purposes), it would be important to investigate in detail whether sensible information is not being sent over the push notification messages by the app, as we discussed in Section 5.

SAPITO provides insights into app libraries and their potential data leakages. Developers can use it to switch to *JADX* for a more detailed evaluation. They can also check privacy policies and terms to understand third-party access to data. The reports can help decide to avoid certain libraries.


Cross-References Information

Cross-references found in the app, in the code of the selected packages. You can use the filter to search for keywords, or click any column header to sort the content of the table. You can download the content of the table as csv or json too.

Directory	Class_called	Element_called	Type	Dangerous
com.onesignal	Lcom/amazon/device/messaging/ADMMessageHandlerBase	<init>	Method	Common
com.onesignal	Landroid/content/Intent	getExtras	Method	Dangerous
com.onesignal	Ljava/lang/StringBuilder	<init>	Method	Common
com.onesignal	Ljava/lang/StringBuilder	append	Method	Common
com.onesignal	Ljava/lang/StringBuilder	toString	Method	Common
com.onesignal	Ljava/lang/String	equals	Method	Common
com.onesignal	Landroid/content/BroadcastReceiver	<init>	Method	Common
com.onesignal	Landroid/app/IntentService	<init>	Method	Common
com.onesignal	Landroid/app/IntentService	setIntentRedelivery	Method	Common
com.onesignal	Lc/m/a/a	completeWakefulIntent	Method	Common
com.onesignal	Landroid/os/AsyncTask	<init>	Method	Common
com.onesignal	Ljava/lang/Object	<init>	Method	Suspicious

Figure 8. xRefs page in SAPITO, where cross-references found in the trackers code are enumerated (cropped image of SAPITO's screen).

7 Conclusion and Further Work

After analyzing data leaks that occurred due to trackers in COVID-19-related apps, we can conclude that sensitive information may have been transferred to the servers of these trackers. This could have happened either directly through the processing of push notifications or indirectly when information was harvested. There is a potential for these tracker companies to infer the COVID-19 status of the app users based on the nature of the information. This privacy concern has affected hundreds of millions of users. Our discussion also touched upon data protection initiatives introduced by the government sector to provide guidelines and requirements for safeguarding the privacy of citizens' data. However, industry and government sectors need to be more adequately informed about the potential impact of these libraries on data protection despite substantial research focused on the Android tracker ecosystem. There is a need for improvement in privacy policies and data protection impact assessments.

We have developed a tool called SAPITO to help privacy analysts investigate data leakages in Android apps through third-party libraries. It can detect the names of libraries and show potentially malicious instructions within their code. During events like the COVID-19 pandemic, SAPITO's reports could have played a crucial role in governments sponsoring and providing apps by enabling proactive detection of privacy issues. However, it is necessary to note that SAPITO currently has certain limitations. While it can identify trackers present in the app, it does not verify whether they are invoked at any point, which can lead to a few false positives. This means a few libraries could be highlighted as trackers, although they are not harmful. These potential false positives are not critical in the context of the tool's expected use: SAPITO flags potentially suspicious code that the analyst should investigate further using more focused tools. More problematic are false negatives, which are codes that leak information but are not detected by the tool. We have developed the rules for detecting dangerous behavior based on the knowledge gained by our team during our research and made them easy to update and extend to reduce the number of potential false negatives. SAPITO doesn't offer a fully automated report of data leaks. A potential enhancement is the addition of a module to construct diagrams and process iOS applications.

Moving forward, we are determined to broaden our research beyond COVID-19 apps and delve into the FinTech application landscape. Our primary goal is to enhance security and privacy by leveraging technology to analyze regulatory compliance. We will explore the role of bank secrecy in certain jurisdictions while using machine learning to personalize FinTech applications and remain compliant with data protection regulations.

Declarations

Authors' Contributions

NS performed the activities described in our methodology, analysed the results, identified the main points detailed in our discussion, and designed and coded SAPITO. GB and JDC provided guidance and guidelines during the research. All authors wrote and reviewed the paper.

Competing interests

The authors declare that they have no competing interests.

References

AGESIC Uruguay (2020). Coronavirus uy. Available at: https://play.google.com/store/apps/details?id=uy.gub.salud.plancovid19uy&hl=es_UY. Accessed: 2023-04-21.

Ahmed, Michelin, Xue, Ruj, Malaney, Kanhere, Seneviratne, Hu, Janicke, and Jha (2020). A survey of covid-19 contact tracing apps. *IEEE Access*, 8:134577–134601. DOI: 10.1109/ACCESS.2020.3010226.

Alfayez, Al-Sinayyid, and AL-Ameri (2021). Mobile applications developed by arab countries in response to covid-19: A review. *Journal of Information System and Technology Management*, 6:200–211. DOI: 10.35631/JISTM.622016.

Ali, ElFadl, Abujazar, Aziz, Abd-Alrazaq, Shah, Brahim, Belhaouari, Househ, and Alam (2020). Contact tracing apps for covid-19: Access permission and user adoption. In *2020 7th International Conference on Behavioural and Social Computing (BESC)*, pages 1–7. DOI: 10.1109/BESC51023.2020.9348327.

Allix, Bissyandé, Klein, and Le_Traon (2016). Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA. ACM. DOI: 10.1145/2901739.2903508.

Android (2021). Instant apps. Available at: https://developer.android.com/topic/google-play-instant Accessed: 2023-04-21.

Android (2023). Notification permission. Available at: https://developer.android.com/develop/ui/views/notifications/notification-permission Accessed: 2023-05-01.

Anglemyer, Moore, Parker, Chambers, Grady, Chiu, Parry, Wilczynska, Flemyng, and Bero (2020). Digital contact tracing technologies in epidemics: a rapid review. *Cochrane Database Syst Rev*. DOI: 10.1002/14651858.CD013699.

AWO Agency (2020). Report on the privacy risks of covid-19 software. Available at: https://www.awo.agency/files/report-on-the-privacy-risks-of-COVID-19-software.pdf.

AWO Agency (2021). Government responses to the covid-19 pandemic. Available at: https://www.awo.agency/files/LSE-government-response-to-the-Covid-19-pandemic.pdf.

Azad, Arshad, Akmal, Riaz, Abdullah, Imran, and Ahmad (2020). A first look at privacy analysis of covid-19 contact-tracing mobile applications. *IEEE Internet Things J*. DOI: 10.1109/JIOT.2020.3024180.

Binns, Lyngs, Kleek, V., Zhao, Libert, and Shadbolt (2018). Third party tracking in the mobile ecosystem. In *Proceedings of the 10th ACM Conference on Web Science*, WebSci '18, page 23–31, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3201064.3201089.

Bock, Kühne, Mühlhoff, Ost, Pohle, and Rehak (2020). Data protection impact assessment for the corona app. *SSRN Electronic Journal*. DOI: 10.2139/ssrn.3588172.

Branch (2022). Branch privacy policy. Available at: https://branch.io/policies/privacy-policy/ Accessed: 2022-09-29.

Caputo, D., Pagano, F., Bottino, G., Verderame, L., and Merlo, A. (2022). You can't always get what you want: Towards user-controlled privacy on android. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1. DOI: 10.1109/TDSC.2022.3146020.

CERN (2013). Zenodo. Available at: https://www.eui.eu/Research/Library/ResearchGuides/Economics/Statistics/DataPortal/Zenodo Accessed: 2022-09-20.

Cho, Ippolito, and Yu (2020). Contact tracing mobile apps for covid-19: Privacy considerations and related trade-offs. *ArXiv*, abs/2003.11511. DOI: 10.48550/arXiv.2003.11511.

Continella, A., Fratantonio, Y., Lindorfer, M., Puccetti, A., Zand, A., Kruegel, C., and Vigna, G. (2017). Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, San Diego, CA. Available at: https://publik.tuwien.ac.at/files/publik_278933.pdf.

CovidSafeBE (2020). Covid safe privacy policy. Available at: https://cert-app.be/en/privacy-covidsafe.html. Accessed: 2022-09-30.

Dehaye and Reardon (2020). Proximity tracing in an ecosystem of surveillance capitalism. In *Proceedings of the 19th Workshop on Privacy in the Electronic Society*, WPES'20, page 191–203, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3411497.3420219.

Desnos (2022). Androguard. Available at: https://github.com/androguard/androguard. Accessed: 2022-09-19.

Dieter, Helmond, Tkacz, Vlist, and Weltevrede (2021). Pandemic platform governance: Mapping the global ecosystem of covid-19 response apps. *Internet Policy Review*, 10. Available at: https://www.econstor.eu/handle/10419/245334.

Dubai Health Authority (2020). Covid19 - dxb smart app privacy policy. Available at: https://www.dha.gov.ae/en/privacy-policy. Accessed: 2022-09-30.

European Data Protection Supervisor (2022). Privacy – a fundamental right. Available at: https://edps.europa.eu/data-protection/data-protection_en. Accessed: 2022-09-30.

European Parliament (2016). Gdpr. Available at: https://eur-lex.europa.eu/eli/reg/2016/679/oj. Accessed: 2024-02-21.

Exodus (2022a). Exodus standalone. Available at: https://github.com/Exodus-Privacy/exodus-standalone. Accessed: 2022-09-20.

Exodus (2022b). Exodus static analysis. Available at: https://exodus-privacy.eu.org/en/post/

exodus_static_analysis/. Accessed: 2022-09-19.

GetSocial (2022). Device fingerprinting. Available at: https://blog.getsocial.im/device-fingerprinting-for-mobile-attribution/. Accessed: 2022-10-03.

Hatamian, Wairimu, Momen, and Fritsch (2021). A privacy and security analysis of early-deployed covid-19 contact tracing android apps. *Empirical Softw. Engg.*, 26(3). DOI: 10.1007/s10664-020-09934-4.

He, Y., Yang, X., Hu, B., and Wang, W. (2019). Dynamic privacy leakage analysis of android third-party libraries. *Journal of Information Security and Applications*, 46:259–270. DOI: 10.1016/j.jisa.2019.03.014.

HSCNI Northern Ireland (2020a). Dpia covidcert. Available at: https://covid-19.hscni.net/covidcert-ni-mobile-app/. Accessed: 2022-09-30.

HSCNI Northern Ireland (2020b). Dpia stopcovid ni. Available at: https://covid-19.hscni.net/wp-content/uploads/2020/10/DPIA-for-StopCOVID-NI-Proximity-App-14.10.pdf. Accessed: 2022-09-30.

HSE Ireland (2020). Dpia covid tracker app. Available at: https://github.com/HSEIreland/covidtracker-documentation/blob/master/documentation/privacy/. Accessed: 2022-09-30.

ICO United Kingdom (2020). Covid-19 contact tracing: data protection expectations on app development. Available at: https://ico.org.uk/media/for-organisations/documents/2617676/ico-contact-tracing-recommendations.pdf. Accessed: 2022-10-04.

jadx (2022). jadx - dex to java decompiler. Available at: https://github.com/skylot/jadx. Accessed: 2022-09-19.

Kollnig, Binns, Dewitte, Kleek, V., Wang, Omeiza, Webb, and Nigel (2021). A fait accompli? an empirical study into the absence of consent to third-party tracking in android apps. *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*. Available at: https://www.usenix.org/conference/soups2021/presentation/kollnig.

Koronavirus app Croatia (2020). Dpia koronavirus. Available at: https://www.koronavirus.hr/uploads/Stop_COVID_19_Data_Protection_Impact_Assesment_Summary_2020_11_16_58dea76816.pdf. Accessed: 2022-09-30.

Kouliaridis, Kambourakis, Chatzoglou, Geneiatakis, and Wang (2021). Dissecting contact tracing apps in the android platform. *PLoS One*. DOI: 10.1371/journal.pone.0258074.

Liu, Liu, Zhu, Wang, and Zhang (2020). Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. *IEEE Transactions on Mobile Computing*, 19(5):1184–1199. DOI: 10.1109/TMC.2019.2903186.

MCIT Indonesia (2020). Pedulilindungi privacy policy. Available at: https://www.pedulilindungi.id/kebijakan-privasi-data?lang=en. Accessed: 2022-09-30.

MIT Technology Review (2020a). Covid tracing tracker. Available at: https://docs.google.com/spreadsheets/d/1ATalASO8KtZMx_zJREoOvFh0nmB-sAqJ1-CjVRSCOw. Accessed: 2022-10-05.

MIT Technology Review (2020b). Mit technology review covid tracing tracker. Available at: https://www.technologyreview.com/2020/12/23/1015557/covid-apps-contact-tracing-suspended-replaced-or-relaunched/. Accessed: 2022-10-03.

MoH New Zealand (2020). Dpia nz covid tracer. Available at: https://www.health.govt.nz/system/files/documents/pages/contact_tracing_app_pia_for_release_10_final.pdf. Accessed: 2022-09-30.

MoH Turkey (2020). Hayat eve sigar privacy policy. Available at: https://hayatevesigar.saglik.gov.tr/gizlilik_politikasi_eng_index_V2.html. Accessed: 2022-09-30.

MoI Qatar (2020). Ehteraz privacy policy. Available at: https://portal.moi.gov.qa/met2/privacyehteraz.html. Accessed: 2022-09-30.

Nakamoto, Wang, Guo, and Zhuang (2020). A qr code–based contact tracing framework for sustainable containment of covid-19: Evaluation of an approach to assist the return to normal activity. *JMIR Mhealth Uhealth*. DOI: 10.2196/22321.

Nesmachnow and Iturriaga (2019). Cluster-uy: Collaborative scientific high performance computing in uruguay. In Torres, M. and Klapp, J., editors, *Supercomputing*, pages 188–202, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-38043-$4_1$6.

NHS United Kingdom (2020). Dpia nhs. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1028998/NHS_COVID_19_App_DPIA.pdf. Accessed: 2022-09-30.

Perera and Hogben (2018). Capillary. Available at: https://github.com/google/capillary. Accessed: 2022-10-02.

Pushwoosh (2019). Pushwoosh push notification privacy. Available at: https://blog.pushwoosh.com/blog/how-to-work-with-sensitive-data-if-you-want-to-use-push-notifications-2/. Accessed: 2022-09-24.

Razaghpanah, Nithyanand, Vallina-Rodriguez, Sundaresan, Allman, Kreibich, and Gill (2018). Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. In *NDSS*. Available at: http://hdl.handle.net/20.500.12761/507.

Samhi, Allix, Bissyandé, and Klein (2021). A first look at android applications in google play related to covid-19. *Empirical Software Engineering*, 26. DOI: 10.1007/s10664-021-09943-x.

Serrano (2023). Sapito gitlab. Available at: https://gitlab.fing.edu.uy/gsi/sapito. Accessed: 2023-11-21.

Serrano, Betarte, and Campo (2023a). Analyzed binaries. Available at: https://gitlab.fing.edu.uy/gsi/trackers-covid/-/blob/main/listado_apps.csv. Accessed: 2023-06-05.

Serrano, Betarte, and Campo (2023b). Analyzed policies. Available at: `https://gitlab.fing.edu.uy/gsi/trackers-covid/-/blob/main/policies.csv`. Accessed: 2023-06-05.

Serrano, N., Betarte, G., and Campo, J. D. (2023c). Third-party trackers in covid-19 mobile applications can enable privacy leaks. In *Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing*, LADC '23, page 80–89, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3615366.3615426.

Startapp (2022). Startapp privacy policy. Available at: `https://www.start.io/policy/privacy-policy-site/`. Accessed: 2022-09-29.

Stevens, Gibler, Crussell, Erickson, and Chen (2012). Investigating user privacy in android ad libraries. *Workshop on Mobile Security Technologies (MoST)*. Available at: `https://web.cs.ucdavis.edu/~hchen/paper/most2012ad.pdf`. Accessed: 2022-09-29.

Tahaei and Vaniea (2021). "developers are responsible": What ad networks tell developers about privacy. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI EA '21, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3411763.3451805.

Tangari, Ikram, Ijaz, Kaafar, and Berkovsky (2021). Mobile health and privacy: Cross sectional study. *BMJ*, 373:n1248. DOI: 10.1136/bmj.n1248.

US Department of Health and Human Services (1996). Hipaa. Available at: `https://www.hhs.gov/hipaa/index.html`. Accessed: 2024-02-21.

Wang, He, Wang, Xia, Li, Wu, Zhou, Luo, Sui, Guo, and Xu (2021a). Beyond the virus: A first look at coronavirus-themed mobile malware.

Wang, J., Xiao, Y., Wang, X., Nan, Y., Xing, L., Liao, X., Dong, J., Serrano, N., Lu, H., Wang, X., and Zhang, Y. (2021b). Understanding malicious cross-library data harvesting on android. In *USENIX Security Symposium*. Available at: `https://www.usenix.org/conference/usenixsecurity21/presentation/wang-jice`.

Wang, L., He, R., Wang, H., Xia, P., Li, Y., Wu, L., Zhou, Y., Luo, X., Sui, Y., Guo, Y., and Xu, G. (2021c). Beyond the virus: a first look at coronavirus-themed android malware. *Empirical Software Engineering*, 26(4). DOI: 10.1007/s10664-021-09974-4.

Wen, Zhao, Lin, Xuan, and Shroff (2020). A study of the privacy of covid-19 contact tracing apps. In Park, N., Sun, K., Foresti, S., Butler, K., and Saxena, N., editors, *Security and Privacy in Communication Networks*, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-63086-$7_1$7.

World Health Organization (2020). Coronavirus disease COVID-19. Available at: `https://www.who.int/health-topics/coronavirus`. Accessed: 2023-05-09.

Wu, Wu, Wang, Ling, and Yang (2016). Efficient fingerprinting-based android device identification with zero-permission identifiers. *IEEE Access*, 4:8073–8083. DOI: 10.1109/ACCESS.2016.2626395.

Yang, Heemsbergen, and Fordyce (2021). Comparative analysis of china's health code, australia's covidsafe and new zealand's covid tracer surveillance apps: a new corona of public health governmentality? *Media International Australia*. DOI: 10.1177/1329878X20968277.

Zhou, Jia, Skinner, Yang, and Claude (2021). Lessons on mobile apps for covid-19 from china. *Journal of Safety Science and Resilience*, 2(2):40–49. DOI: 0.1016/j.jnlssr.2021.04.002.

A SAPITO Pseudocode

In this Appendix, we provide pseudocode for the main functions of SAPITO.

Initialization:

```
APK_analysis = ANDROGUARD.analyze_APK(path_to_APK)
```

Main Report():

```
suspicious_libs = {}
suspicious_libs.add(xref_analysis(all_libraries))
suspicious_libs.add(permissions_analysis(all_libraries))
suspicious_libs.add(root_analysis(all_libraries))
suspicious_libs.add(reflection_analysis(all_libraries))
suspicious_libs.add(connections_analysis(all_libraries))
suspicious_libs.add(notifications_analysis(all_libraries))
for lib in suspicious_libs:
    scores[lib] = (lib.has_xrefs()∗10 + lib.has_notifications()∗7 +
        lib.has_permissions()∗5 + lib.has_root()∗3 + lib.
        has_reflections()∗2) ∗ lib.has_connections()
return ordered(scores)
```

xref_analysis(libraries_to_analyze):

```
dangerous_xrefs = {}
classes = APK_analysis.get_classes()
for clss in classes:
    if clss.get_library() in libraries_to_analyze:
        for method in clss.get_methods():
            for instruction in method.get_source_code():
                #dangerous_calls_list loaded from JSON file
                if instruction in dangerous_calls_list:
                    dangerous_xrefs.add(instruction)
return dangerous_xrefs
```

permissions_analysis(libraries_to_analyze):

```
permissions_checks = {}
classes = APK_analysis.get_classes()
for clss in classes:
    if clss.get_library() in libraries_to_analyze:
        for method in clss.get_methods():
            for instruction in method.get_source_code():
                if checking_for_permissions(instruction):
                    permissions_checks.add(instruction)
return permissions_checks
```

root_analysis(libraries_to_analyze):

```
root_checks = {}
classes = APK_analysis.get_classes()
#root_check_rules loaded from JSON file
for rule in root_check_rules:
    string_analysis = APK_analysis.get_strings(rule)
    for method in string_analysis.get_xrefs_from():
        if method.get_class() in libraries_to_analyze:
            root_checks.add(method)
return root_checks
```

reflection_analysis(libraries_to_analyze):

```
reflection_calls = {}
for method in APK_analysis_get_all_methods():
    for calling_method in method.get_xrefs_from():
        if calling_method.get_class() in libraries_to_analyze:
            reflection_calls.append(get_reflection_calls(
                    calling_method.get_source_code())
return reflection_calls
```

connections_analysis(libraries_to_analyze):

```
connection_calls = {}
classes = APK_analysis.get_classes()
for clss in classes:
    if clss.get_library() in libraries_to_analyze:
        for method in clss.get_methods():
            for instruction in method.get_source_code():
                #connection_calls_list loaded from JSON file
                if instruction in connectionc_alls_list:
                    connection_calls.add(instruction)
return connection_calls
```

push_notification_analysis(libraries_to_analyze):

```
push_notification_calls = {}
classes = APK_analysis.get_classes()
#push_notificaction_classes loaded from JSON file
for pn_class in push_notificaction_classes:
    for method in APK_analysis.get_class_methods(pn_class):
        for call in method.get_xrefs_from():
            if call.get_class() in libraries_to_analyze:
                push_notification_calls.add(call)
return push_notification_calls
```