



UNIVERSIDAD DE LA REPÚBLICA DEL URUGUAY

MASTER THESIS

**Detection and classification of privacy leaks enabled by
third-party trackers in COVID-19 mobile applications**

Author:

Ing. Nicolás SERRANO

Supervisors:

Dr. Gustavo BETARTE
Dr. Juan Diego CAMPO

*A thesis submitted in fulfillment of the requirements
for the degree of Magister en Informática*

April 29, 2024

Declaration of Authorship

I, Ing. Nicolás SERRANO, declare that this thesis titled, “Detection and classification of privacy leaks enabled by third-party trackers in COVID-19 mobile applications” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSIDAD DE LA REPÚBLICA DEL URUGUAY

Abstract

PEDECIBA
Área Informática

Magister en Informática

**Detection and classification of privacy leaks enabled by third-party trackers in
COVID-19 mobile applications**

by Ing. Nicolás SERRANO

Abstract

Since 2019, the world has been experiencing a pandemic without precedents in our current technological era. Governments and other high-profile organizations devoted special efforts to developing and sponsoring mobile applications that, while varying in their goals, tried to help contain the spread of COVID-19 and enable people to have the best quality of life possible. However, while third-party libraries and their impact on the user's privacy have been studied before, especially those considered trackers, these have found their way into COVID-19 applications backed by high-profile organizations. By trackers we considered third-party libraries included in applications to provide certain functionalities that, in addition, gather information regarding the application, the device and their use, and send it to their servers. The research for this thesis found that 402 out of 595 studied applications contained at least one tracker. In addition, it was confirmed that sensitive information was transferred to the tracker servers, potentially disclosing the health status of the application users. On the other hand, evidence indicates that governments can improve their data protection impact assessments and the disclosure they make in their privacy policies; the latter also applies to trackers. Finally, SAPITO, an easy-to-use open-source tool, is presented. Based on the knowledge and lessons learned during this research, it was created with the objective of helping privacy teams and researchers to detect automatically data leakages when analyzing third-party libraries in Android applications.

Keywords: Privacy, Trackers, SDK, Android, Mobile Applications, COVID-19.

UNIVERSIDAD DE LA REPÚBLICA DEL URUGUAY

Resumen

PEDECIBA
Área Informática

Magister en Informática

Detección y clasificación de violaciones en la privacidad de datos en aplicaciones móviles para la COVID-19 debido a la inclusión de librerías de terceros en estas

por Ing. Nicolás SERRANO

Resumen

Desde el 2019, el mundo ha venido sufriendo una pandemia sin precedentes en la presente era tecnológica. Gobiernos y otras organizaciones de alto perfil han destinado recursos especialmente para el desarrollo y promoción de aplicaciones móviles que, aunque variando en su objetivo, estuvieron enfocadas a contener el avance de la COVID-19, permitiendo a los ciudadanos poder tener la mejor calidad de vida posible durante la pandemia. Sin embargo, a pesar de que la utilización de librerías de terceros y el impacto que esto tiene en la privacidad de los usuarios ha sido estudiado previamente, en especial cuando estas librerías son en efecto “trackers”, estas fueron incluidas en las aplicaciones móviles usadas para combatir la COVID-19. Con “tracker” hacemos referencia a librerías de terceros que proveen ciertas funcionalidades al ser incluidas en aplicaciones móviles y que, además, recolectan información en tiempo real sobre la aplicación, el dispositivo y las interacciones del usuario, enviando luego esta información recogida a sus servidores. En la investigación comprendida en esta tesis, encontramos que 402 de las 595 aplicaciones móviles estudiadas contenían al menos un “tracker”. Adicionalmente, identificamos que información sensible fue transferida a los servidores de los “trackers”, potencialmente revelando información sobre el estado de salud de los usuarios de estas aplicaciones. Por otro lado, sobre lo investigado para estas aplicaciones móviles, la evidencia indica que los gobiernos podrían mejorar sus evaluaciones de impacto en protección de datos y lo que es detallado en sus políticas de privacidad; esto último también aplicando a los proveedores de “trackers”. Por último, presentamos SAPITO, una herramienta para el análisis de privacidad open-source con foco en su facilidad de uso. En base al conocimiento adquirido y las lecciones aprendidas durante nuestra investigación, SAPITO fue creado con el objetivo de apoyar a los equipos e investigadores de privacidad a detectar de manera automática problemas de privacidad al analizar el uso de librerías de terceros en aplicaciones móviles para Android.

Acknowledgements

First and foremost, I wish to extend my deepest gratitude to my Thesis Supervisors, Gustavo Betarte (also serving as Academic Director) and Juan Diego Campo. Their unwavering support, insightful guidance, steadfast dedication, trust, and patience were not only vital in the completion of this thesis but also instrumental in navigating the extensive research endeavors associated with my pursuit of a Master's Degree.

Additionally, I would like to express my sincere appreciation to the esteemed panel consisting of Federico La Rocca (as Reviewer), Álvaro Martín (in the capacity of President), and Lorena Etcheverry for graciously evaluating this work.

Lastly, I am profoundly thankful to my wife for her continual encouragement and support, and to my family for instilling in me the motivation from an early age to embark upon such stimulating academic pursuits.

Contents

Declaration of Authorship	iii
Abstract	v
Keywords	vi
Resumen	vii
Acknowledgements	ix
1 Introduction	1
2 Related Work	7
3 Methodology and Tools	9
3.1 On the Hunt for Trackers	9
3.1.1 Coronavirus Apps From Around the World	11
3.1.2 Preparing the Data Set Using ClusterUY	11
3.1.3 Trackers in Sight. Using Exodus as Detection Tool	11
3.1.4 Trackers Autopsy: Dissecting Apps to Detect Data Harvesting	12
3.1.5 Limitations Faced	12
3.2 Research Ethics	13
4 Findings	15
4.1 What do the Numbers Say: Applications and Trackers Statistics	15
4.2 Trackers in Action: Observed Methodology of Trackers	18
4.2.1 Real-Time vs Store & Retrieve:	18
4.2.2 Event-Based vs General:	20
4.2.3 Centralized vs Iterative vs Decentralized:	20
4.2.4 Centralized Connections vs Decentralized Connections:	20
4.3 Detailed Information of Analyzed Trackers	20
4.3.1 Airship (Centralized Connection)	21
Introduction	21
Tracked Information	21
Data Flow Diagram	23
4.3.2 Amplitude (Store and Retrieve Collection)	23
Introduction	23

	Tracked Information	23
	Data Flow Diagram	25
4.3.3	Branch (Real-Time Collection)	25
	Introduction	25
	Tracked Information	25
	Data Flow Diagram	27
4.3.4	Bugsnag (Decentralized Connection)	27
	Introduction	27
	Tracked Information	27
	Data Flow Diagram	29
4.3.5	Flurry (Decentralized Collection)	29
	Introduction	29
	Tracked Information	29
	Data Flow Diagram	31
4.3.6	Google AdMob (Iterative Collection)	31
	Introduction	31
	Tracked Information	31
	Data Flow Diagram	33
4.3.7	Mapbox (Event-Based Collection)	33
	Introduction	33
	Tracked Information	33
	Data Flow Diagram	35
4.3.8	Matomo (General Collection)	35
	Introduction	35
	Tracked Information	35
	Data Flow Diagram	37
4.3.9	OneSignal (Centralized Collection)	37
	Introduction	37
	Tracked Information	37
	Data Flow Diagram	39
4.4	Tracking the Trackers Tracks: On Data Collection	39
4.5	Leaking Data in Every Message: On Push Notifications	43
5	SAPITO: a tool for information leaking analysis of Android mobile applica-	
	tions	45
5.1	Analysing Apps One Hop at a Time: Presenting SAPITO	45
5.1.1	Motivation for Developing the Tool	45
5.1.2	Technology Behind SAPITO	45
5.1.3	What is SAPITO Capable of?	46
5.2	SAPITO Features in Detail	49
5.2.1	APK Loading	49
5.2.2	Main Report	50

5.2.3	Packages Clustering	51
5.2.4	App Info	52
5.2.5	Google Play Info	53
5.2.6	Exodus Info	54
5.2.7	Library Cross-References	55
5.2.8	Library Permissions Checks	56
5.2.9	Library Rooted Checks	57
5.2.10	Library Reflection Use	58
5.2.11	Library Connections	59
5.2.12	Library Push Notifications Use	60
5.2.13	Ruleset Loading	61
6	Discussion	65
6.1	On the Known Practices of Device Identification: Impact of Data Collection	65
6.2	A Design Issue: Impact of Using Push Notification Services	66
6.3	Data Privacy Posture from Governments	66
6.4	What the Trackers Declare	67
6.5	Observations Regarding Possible Roots of the Problem	68
7	Conclusion and Further Work	71
	Bibliography	75
A	List of COVID-19 Android applications analyzed	83
B	Detailed Information of Other Analyzed Trackers	89
B.1	AdColony	89
B.1.1	Introduction	89
B.1.2	Tracked Information	89
B.1.3	Data Flow Diagram	91
B.2	AltBeacon	91
B.2.1	Introduction	91
B.2.2	Tracked Information	91
B.2.3	Data Flow Diagram	92
B.3	AppNext	92
B.3.1	Introduction	92
B.3.2	Tracked Information	92
B.3.3	Data Flow Diagram	94
B.4	Braze	94
B.4.1	Introduction	94
B.4.2	Tracked Information	94
B.4.3	Data Flow Diagram	96
B.5	Google Firebase Analytics	96
B.5.1	Introduction	96

B.5.2	Tracked Information	97
B.5.3	Data Flow Diagram	98
B.6	Google Tag Manager	98
B.6.1	Introduction	98
B.6.2	Tracked Information	98
B.6.3	Data Flow Diagram	98
B.7	New Relic	99
B.7.1	Introduction	99
B.7.2	Tracked Information	99
B.7.3	Data Flow Diagram	100
B.8	Open Telemetry	100
B.8.1	Introduction	100
B.8.2	Tracked Information	100
B.8.3	Data Flow Diagram	100
B.9	Pushwoosh	101
B.9.1	Introduction	101
B.9.2	Tracked Information	101
B.9.3	Data Flow Diagram	102
B.10	Segment	102
B.10.1	Introduction	102
B.10.2	Tracked Information	102
B.10.3	Data Flow Diagram	104
B.11	Splunk MINT	104
B.11.1	Introduction	104
B.11.2	Tracked Information	104
B.11.3	Data Flow Diagram	106
B.12	Startapp	106
B.12.1	Introduction	106
B.12.2	Tracked Information	106
B.12.3	Data Flow Diagram	109
C	Examples of Trackers Code	111

List of Figures

1.1	COVID-19 statistics for Uruguay. Source Google Statistics, with data from: JHU CSSE COVID-19 Data.	1
1.2	Data tracking scenario	4
3.1	Tracker analysis methodology.	10
4.1	Proportion of concurrent usage of trackers.	18
4.2	Data flow in Airship	23
4.3	Data flow in Amplitude	25
4.4	Data flow in Branch	27
4.5	Data flow in Bugsnag	29
4.6	Data flow in Flurry	31
4.7	Data flow in Google AdMob	33
4.8	Data flow in Mapbox	35
4.9	Data flow in Matomo	37
4.10	Data flow in OneSignal	39
4.11	Tracker data collection example.	40
4.12	Tracker push notification example	44
5.1	SAPITO's architecture	46
5.2	Landing page in SAPITO, where the user must select a binary to analyze.	49
5.3	Report page in SAPITO, where potential trackers are highlighted.	50
5.4	Cluster page in SAPITO, where trackers are grouped to detect similarities between them.	51
5.5	App info page in SAPITO, where general information of the app is presented.	52
5.6	Store info page in SAPITO, where information extracted from Google Play is detailed.	53
5.7	Exodus info page in SAPITO, where privacy reports related to the tracker from Exodus are shown.	54
5.8	xRefs page in SAPITO, where cross-references found in the trackers code are enumerated.	55
5.9	Permissions page in SAPITO, where permissions checks found in the trackers code are enumerated.	56
5.10	Rooted page in SAPITO, where rooted and similar checks found in the trackers code are enumerated.	57

5.11	Reflection page in SAPITO, where reflection calls found in the trackers code are enumerated.	58
5.12	Connections page in SAPITO, where connections calls found in the trackers code are enumerated.	59
5.13	Notifications page in SAPITO, where push notification usage found in the trackers code are enumerated.	60
B.1	Data flow in AdColony	91
B.2	Data flow in AppNext	94
B.3	Data flow in Braze	96
B.4	Data flow in Google Firebase Analytics	98
B.5	Data flow in New Relic	100
B.6	Data flow in Pushwoosh	102
B.7	Data flow in Segment	104
B.8	Data flow in Splunk MINT	106
B.9	Data flow in Startapp	109

List of Tables

1.1	Example of analyzed apps and their purposes.	3
4.1	Number of trackers found in the studied applications, and their respective percentage.	15
4.2	Top 15 applications with most trackers included.	16
4.3	Increase and decrease of trackers per application, based on <i>Exodus</i> historical reports.	16
4.4	Number of applications (out of 595) in which the identified trackers were found, together with their percentage of occurrence.	17
4.5	Types of services offered to the developers by the trackers analyzed. Their main service was chosen for trackers that fit more than one category.	19
4.6	For each service category, count of times trackers belonging to these categories were included.	19
4.7	Tracked information by Airship.	22
4.8	Tracked information by Amplitude.	24
4.9	Tracked information by Branch.	26
4.10	Tracked information by Bugsnag.	28
4.11	Tracked information by Flurry.	30
4.12	Tracked information by Google Ad Mob.	32
4.13	Tracked information by Mapbox.	34
4.14	Tracked information by Matomo.	36
4.15	Tracked information by One Signal.	38
4.16	Applications that were analyzed to detect tracking behavior.	41
4.17	Type of information harvested by the trackers and sent to their servers.	43
6.1	Trackers declaration of data collected with their Android SDK.	69
A.1	List of studied COVID-19 Android applications (Part 1/5).	83
A.2	List of studied COVID-19 Android applications (Part 2/5).	84
A.3	List of studied COVID-19 Android applications (Part 3/5).	85
A.4	List of studied COVID-19 Android applications (Part 4/5).	86
A.5	List of studied COVID-19 Android applications (Part 5/5).	87
B.1	Tracked information by AdColony.	90
B.2	Tracked information by AppNext.	93

B.3	Tracked information by Braze.	95
B.4	Tracked information by Google Firebase Analytics.	97
B.5	Tracked information by New Relic.	99
B.6	Tracked information by Pushwoosh.	101
B.7	Tracked information by Segment.	103
B.8	Tracked information by Splunk MINT.	105
B.9	Tracked information by Startapp. (Part 1/2)	107
B.10	Tracked information by Startapp. (Part 2/2)	108

Chapter 1

Introduction

At this point, sadly, Coronavirus (COVID-19) does not need a lengthy introduction. It is an infectious disease caused by the SARS-CoV-2 virus, where infected people experience mild to moderate respiratory illness [30]. The first case of COVID-19 was reported at the end of 2019, and it rapidly became a global pandemic. At the time of writing this thesis, there have been almost 700 millions infections, and around 6.8 millions deaths caused by this virus [34]. While the weekly deaths counter is lower than during the first two years of the disease, it is still a severe global health issue.

Technology permeates every aspect of our life. One field experiencing a continuous expansion in recent years is digital health. It includes mobile health, health information technology, wearable devices, telehealth and telemedicine, and personalized medicine. In the context of COVID-19, mobile applications (apps from now on) for digital health have been seen in almost every country. Some sponsors of the development of that technology have been governments, international organizations, health institutions, and universities. The motivation for this work started at the beginning of 2022, when Uruguay was traversing its most difficult moment regarding the impact of COVID-19, as shown in Figure 1.1. The overall objective for this thesis was to understand the use of third-party libraries in the context of COVID-19 Android apps, and measure its impact from the perspective of the users' privacy. In table 1.1, some examples of analyzed apps are listed (apps marked with '*' symbol have other purposes besides those stated in the table).

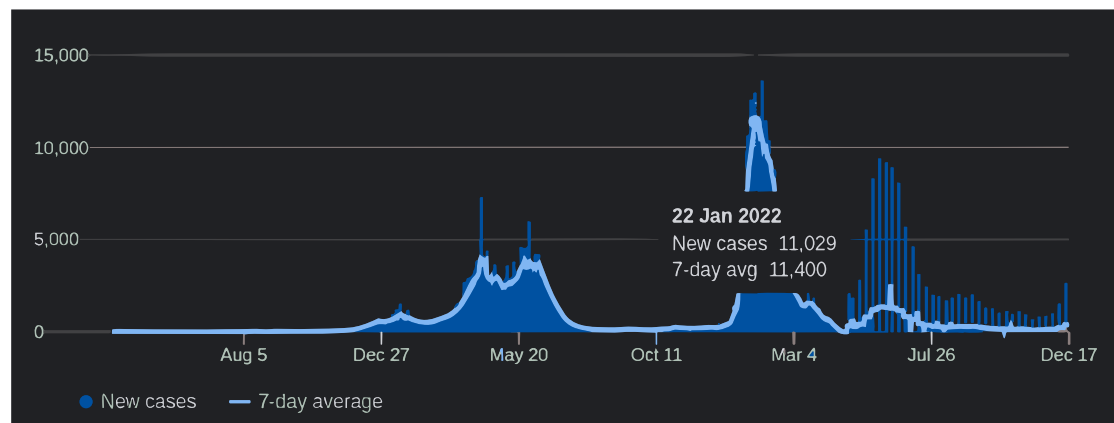


FIGURE 1.1: COVID-19 statistics for Uruguay. Source Google Statistics, with data from: JHU CSSE COVID-19 Data.

The study of this particular class of applications is motivated by the fact that the intended user usually trusts the institutions sponsoring them. Furthermore, in some cases, governmental apps were mandatory or convenient to continue with people's daily activities (as in the cases where, to assist a public place, vaccination certificates stored in the application needed to be shown). Data protection and privacy have been significant concerns related to COVID-19 apps, especially those whose purpose is contact tracing, since it involves the identification of citizens, what places they frequent, who they meet, and sensitive health information regarding their possible COVID-19 infection. Privacy by design is a must, and special care is required to comply with regulations like HIPAA, GDPR, and other applicable data protection acts and laws. Moreover, data protection impact assessments are becoming increasingly used in the development life-cycle of apps of this magnitude. Privacy policies are another critical component. Users must be informed, and give their consent to anything the application, the collector, and the processor do with their data. On the technical side, some developers published the application code as open-source, which enormously contributes to transparency. Transparency and privacy need to go hand in hand.

It is a common practice today, however, for mobile applications to include third-party libraries through software development kits, or SDKs, given the benefits they provide to developers (special functions like a map or login with social media credentials, monetization, crash reports, user engagement, among others). Some libraries collect sensitive information from the user, the device, and how the former interacts with the latter. Information they are capable of tracking includes, among others, *location, device information, application attributes, and app usage times*. This information is later used to create profiles of the app's users, which are then used, for instance, for targeted ads.

In general, the life-cycle of data tracking follows this pattern: first, the company behind the SDK develops it and sets up the infrastructure for the back-end. After that, the company offers its SDK and the app developers start integrating it within their apps code. Later, these developers publish their apps and, at a given time, the users start installing them. This is the moment when the data harvesting starts to happen. As the users use the app, the trackers will start harvesting data from the device, the app, the app's usage, and the user. Later on, based on time or specific events, the trackers will start exfiltrating the harvested data, sending it to their back-end servers. After that, the harvested data is in the hands of the company that provided the SDK, and it could make a legitimate or an illegitimate use of it. The general idea of this process is shown in Figure 1.2.

Another scenario of data leakage involving third-parties SDKs happens when these libraries are used at some point during the process of sending push-notifications to the users. For example, using a third-party infrastructure to deliver sensitive clear text content of push notifications for apps. In that scenario, the privacy of the notification text would be lost.

Given this scenario, while for tracker libraries any piece of information they can obtain from users is precious, for the actual processor of COVID-19-related sensitive

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 Government. 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 Mgmt.	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 1.1: Example of analyzed apps and their purposes.

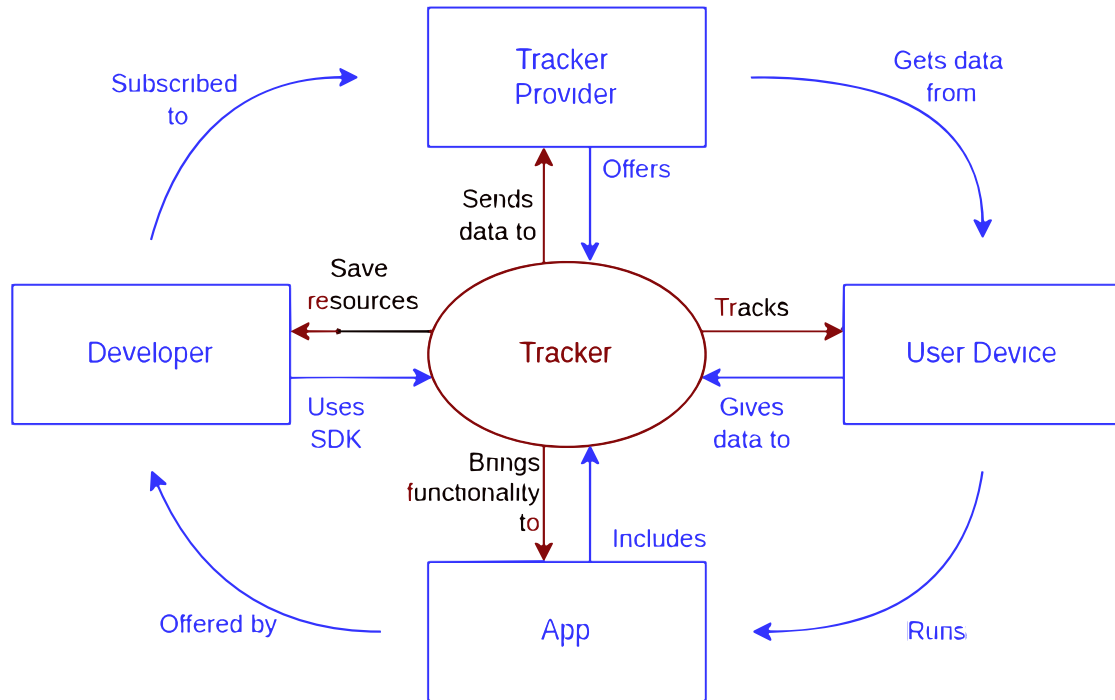


FIGURE 1.2: Data tracking scenario

information it demands great responsibility. Hence, data collected and shared with third parties must be limited to the minimum required and **for the specific purpose of the application**. Many countries were aware of this when developing their contact tracing apps, as stated in [33]: “13. *The use of coding libraries, frameworks, APIs, SDKs, and other software components must be clarified, including those within the mobile operating system. Data collection by third parties for other purposes must be avoided.*”

In this study, the presence of trackers was analyzed in over 500 Android apps related to COVID-19, coming from respected entities such as governments and international organizations. The information they collected was identified, and the impact on the application users was reviewed. In particular, this research tried to answer the following questions:

- RQ1) Are trackers being used in the COVID-19 app ecosystem?
- RQ2) What information is being tracked?
- RQ3) What are the potential impacts on users if trackers are used and harvest that information?

The main contributions of this work are:

- The work for this thesis involved the analysis of almost 600 Android apps related to COVID-19, being the first one to include such number of apps in a privacy-focused research regarding COVID-19. In addition, these apps were carefully identified and selected for our study, since our goal was to study the approach to app privacy taken by governments and other highly regarded organizations.

- We found that over two-thirds of the studied apps included, at least, one tracker. Furthermore, the analysis conducted in this research provided evidence that indicates that sensitive information related to the users and their health may have potentially reached the servers of third-party libraries detected in the apps.
- Privacy-oriented initiatives taken by different governments are presented and discussed. There are indications that the presence of trackers in sensitive applications has not been identified yet as a serious threat to the privacy of citizens' data.
- An open-source tool that helps privacy teams audit third-party libraries included in Android apps was developed. It aims to detect data leakages through them. Particular emphasis was put on its usability so that non-technical analysts could use it.
- Our research was presented in two venues. A brief virtual presentation was prepared for *Congreso interdisciplinario COVID 19, pandemia y pospandemia 2022 (Montevideo, Uruguay)* under the title *Privacidad en aplicaciones móviles: riesgos en el uso de librerías de terceros*. On the other hand, a research paper was presented at *The 12th Latin-American Symposium on Dependable and Secure Computing, (LADC 2023 - La Paz, Bolivia)*, with the title *Third-Party Trackers in COVID-19 Mobile Applications Can Enable Privacy Leaks*. **Our work was awarded the best paper of the conference.** Additionally, an extended version of our work was invited to be presented in a special issue of JISA (Journal of Internet Services and Applications).

The rest of this thesis is structured as follows: Chapter 2 describes research related to this study, highlighting studies of COVID-19 apps, and the trackers' ecosystem. In Chapter 3, the methodology used is described, including its limitations and ethics. Next, Chapter 4 details the direct outcomes of this study, including elaborated information regarding the studied trackers, their methodology of data harvesting, the sensitive data that was collected from the users and their devices, and the problems related to the use of push notifications to send sensitive data to the users using third-party infrastructures. Following that, Chapter 5 describes SAPITO, the automated tool created with the objective of facilitating privacy investigations, based on the experience and knowledge obtained during our analytical research of trackers code. On the other hand, Chapter 6 elaborates on our findings, providing analyzes on the impact of the data collection and push notification issues found previously. Moreover, the stance of governments and tracking companies is described, and further arguments are presented in relation to the causes of these problems, accompanied by general solutions and suggestions based on what was seen during the study. At the end of this chapter, the research questions are discussed as well. The last Chapter 7 briefly details the conclusion of this thesis work, alongside potential future work.

Chapter 2

Related Work

Plenty of research related to COVID-19 apps from different perspectives already exists. For example: apps survey [7], the platform governance [62], their effectiveness [1], their taxonomy [57], new approaches for contact tracing [55], COVID-19 themed malware [90], surveillance [94], and overall lessons learned from the apps [96].

Academic research and industry studies on privacy in these apps have been performed since the beginning of the pandemic [52, 92, 63, 10]. However, most of them did not focus on the impact of SDKs inclusion. This could be dangerous, since a critical perspective of data privacy, sharing personal information with third parties, remained potentially unseen. Moreover, too much focus was given to application permissions. Although this could be dangerous since trackers could piggyback these permissions, it is not necessarily alarming that an application that uses *Bluetooth* as its underlying technology for contact tracing requests Bluetooth-related permissions; or one that performs video calls with a doctor requests *Camera* permissions. On the other hand, the few studies that discussed the use of trackers limited their scope to naming the trackers included [88, 6]. Finally, Dehaye et al. in [36] demonstrated using dynamic analysis how easily third-party libraries can access device private data of apps, and linked it to the potential it may have with contact tracing apps used for COVID-19 containment. We reached the same conclusion in our study using static analysis of the trackers code. Therefore, with the detailed analysis provided by this study of SDKs and their data tracking, we provide a complementary approach to other academic research, adding an essential perspective in relation to privacy in COVID-19 apps.

Outside COVID-19, the intersection of apps, trackers, and privacy has also been studied. Several privacy leak detection tools at the application level have been presented. For example, Continella et al. described *AGRIGENTO* in [29], which outperformed previous tools. Another dynamic privacy leakage analysis for third-parties trackers was studied by He et al. [54]. However, since for this research it was desired to find exactly which library within the app was leaking private information, and in order to avoid false positives and negatives, it was preferred to perform manual static analysis of the selected apps code using specific reverse engineering techniques and tools for that type of analyzes. Additionally, the lack of open-source and well-documented tools geared to the identification of data exfiltration in third-party libraries was one motivation to develop *SAPITO*, as it is discussed in Chapter 5.

The ecosystem of tracking companies was studied by Razaghpanah et al. [78] and Binns et al. [18]. The organizations behind the most used trackers, described there as *Advertising and Tracking Services (ATS)*, found in these studies (Alphabet, Facebook and Microsoft) coincide with the findings in this study scoped to COVID-19 apps. In addition, other ATS highlighted in these studies were also found in this research (AppsFlyer, AdColony, Adobe, Appnext, ComScore, Flurry, Lotame, MixPanel, New Relic, Segment and Startapp). Liu et al. [61] performed a privacy risk analysis focusing on analytic libraries for Android (instead of advertisement libraries, where most research has been made), finding that apps leak private information through these types of libraries as well. In our study, we analyzed 10 of the 26 tracker libraries to validate their tracking behavior. These libraries collected and transmitted user and mobile information. The same was found in advertisement libraries for Android by Stevens et al. [84] and several other studies, whereas in our study, from the five ad libraries that were flagged as trackers, we analyzed three, finding tracking behavior in two of them. The lack of consent from users for data gathering by third-party trackers was studied by Kollnig et al. [59]. In our work, we detected cases where the declaration of data collection made by companies behind trackers differed from the actual data being collected (in all cases they tracked more data than declared), therefore, any potential consent from the users would still be inappropriate for the actual data harvesting that ends being done by tracking libraries. Recently, Caputo et al. [26], besides studying the impact of mobile analytic libraries, proposed *MobHide*, a data anonymization tool, that, although it presented some limitations, it could have the potential to block data harvesting activities by third parties. A focused study on mobile health apps, including what the trackers track, was performed by Tangari et al. [87]. Given the results of our work, it was concluded that, as a subset of mobile health apps, COVID-19 apps shared the same lack of protection regarding the sharing of private information with third parties, even though in general, these apps were backed by governments and national agencies. Finally, the unfavorable position where developers find themselves when deciding on the convenience of using trackers was studied by Tahaei et al. [68]. Given the pressure of launching COVID-19 apps amid a pandemic, it can be suspected that developing teams faced the same issue with the studied apps.

Chapter 3

Methodology and Tools

This chapter describes and discusses methodological aspects that guided our research.

3.1 On the Hunt for Trackers

This section describes the approach followed to detect the trackers included in COVID-19-related apps and the data they harvested from the device, application, and usage. Figure 3.1 illustrates the methodology approach that was utilized.

The term *tracker* is used as it has been used in related works: to describe a third-party library included in apps to provide analytics, advertisements, technical, or other functionalities without the need for developers to code them when this library gathers information regarding the application, the device, and their use, and sends it to its servers.

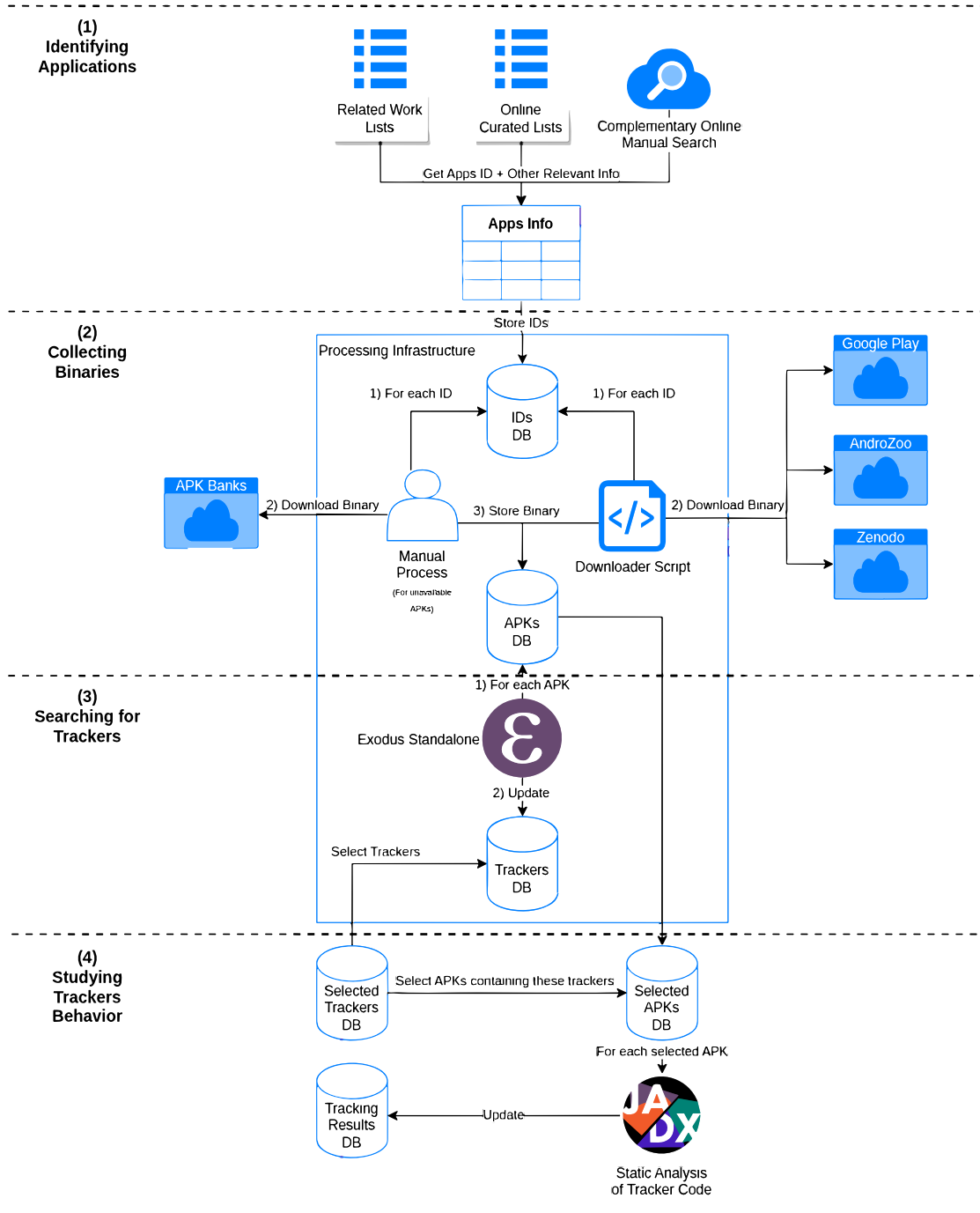


FIGURE 3.1: Tracker analysis methodology.

3.1.1 Coronavirus Apps From Around the World

The first goal of this research was to identify apps related to COVID-19 that were developed or sponsored by government bodies, well-respected international organizations, or recognized universities. Every applicable country, region, or independent jurisdiction was analyzed for neutrality and completeness. This work was done from March 1st to March 15th, 2022.

The study started the identification of apps using related works as reference, such as [5, 28, 9] and curated online lists like [66, 32], and then complemented these with a methodical manual online search (with the goal of finding applications for all relevant countries and regions).

Special attention was placed on the validation of the apps included in our study. It was manually verified that every app selected was indeed used in the context of the COVID-19 pandemic (in contrast to other automatic crawling methods using some keywords, where, for example, the keyword “corona” may yield results totally unrelated to COVID-19).

A total of 619 Android apps were identified. Based on Google Play download figures, 80 were downloaded between 1 million and 5 million times, 14 between 5 million and 10 million times, 21 between 10 million and 50 million, one downloaded in the range 50-100 million, and another one over 100 million times.

3.1.2 Preparing the Data Set Using ClusterUY

The collection, processing, and storage of binaries (APK files) were performed using the infrastructure of ClusterUY [69].

Our primary source of binaries was *Google Play Store*. The number of binaries downloaded from this source was maximized to ensure the integrity of the files. We used the *gplaydl* Python script [51] to automatize the downloading process. However, some apps were unavailable for download in the Uruguayan region or were no longer in the store. Therefore, three additional sources were inspected. *AndroZoo*, a collection of Android apps maintained by Université du Luxembourg [11], was used.

Another data set used was created by Wang et al. [91] for a previous study of malware in COVID-19 apps, hosted in the open data repository *Zenodo* [95]. Finally, some binaries were not available in any of the previous sources. Hence, we decided to download them manually from APK banks such as *Apksos*, *Apkpure* and *Apkgk*. From the initial 619 apps identified, a total of 595 binaries were collected between the 15th and the 30th of March: 356 from *Google Play*, 152 from *AndroZoo*, 8 from *Zenodo*, and 79 manually downloaded. The list of collected binaries can be found in Appendix A.

3.1.3 Trackers in Sight. Using Exodus as Detection Tool

To detect trackers present in the collected binaries, the *exodus-standalone tool* [45], developed by the *Exodus* team, was used. This tool takes an APK file as input and generates an output, including the trackers found in the binary.

The process performed by the tool to detect trackers in the binary relies on signature matching. It dumps the Java classes from the APK file and then, for each class found, compares their package name with a database of previously identified trackers [46].

Setting the tool output as JSON, and with an automation process, the data set of trackers found in the analyzed apps was created. A total of 58 trackers were detected.

3.1.4 Trackers Autopsy: Dissecting Apps to Detect Data Harvesting

Finally, since we wanted to go beyond any other study concerning precisely what data was collected by the trackers in the COVID-19 apps ecosystem, reverse engineering of the trackers code was done. *JADX* tool was used to help with the reverse engineering activities and the static analyses [58]. 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.

To be more specific, the tool helped to identify the exact classes and methods in the APK file where the data was harvested and, following the data flow through the observed tracker package, to detect the internet connection instructions where the collected data was sent to their servers. The apps binaries were not run, thus, no dynamic analyses were performed.

Out of the 58 trackers found, the behaviors of 22 of them were analyzed using the binaries of apps where they were included. Since it was a complex and time-consuming task, and given time and resource restrictions, an analysis of the totality of the trackers flagged by *Exodus* was unfeasible.

After studying the selected trackers, patterns were sought, starting from particular cases and converging to general observed practices. The findings are detailed in Chapter 4.

3.1.5 Limitations Faced

The research faced some limitations. For example, given the lack of tools that provide automated support to perform behavioral analysis on Android apps, most of the inspections were performed manually. The study of all the trackers flagged by *Exodus* was out of the scope of the investigation. Therefore, some trackers may have passed unnoticed (thus, the motivation to develop the inspection tool described in Chapter 5).

It is important to note that although a tracker is present in the application binary, it might not be called and executed (though this is rather unlikely). This possibility was not considered; thus, if a tracker was present in the code, it was assumed it would be executed and the data collected.

Both of the above limitations (analyzing every flagged tracker and trackers passing undetected) could also have been present if we had used dynamic analysis of the application code.

Finally, if a tracker wanted to obfuscate its code using reflection and then dynamically download the strings used in the class and method calls, this study would probably be unfeasible. This specific problem was not found in the analyzed trackers, however.

3.2 Research Ethics

The research has been conducted following rigorous ethic procedures. To understand the behavior of the trackers and detect what information was being collected, it was needed to perform reverse engineering steps of the app binaries (APK files). We limited the scope of these activities to identifying what data from the user, the device, and app usage were harvested and then sent to the tracker servers. No other action was done with the binaries.

All platforms used, like *Google Play* and *Exodus* websites, were accessed according to their terms of service.

Every country, region, and jurisdiction possible was covered, and every application that could be downloaded was analyzed.

Chapter 4

Findings

In this chapter, the results of our research are presented. Details of the analyzed trackers are provided, and there is a discussion about the privacy issues caused by the use of these libraries when implementing COVID-19-related applications.

4.1 What do the Numbers Say: Applications and Trackers Statistics

Over two-thirds of the studied apps included a tracker. It can be appreciated in Table 4.1. Around half of them only contain one tracker, and around 80% of the apps include no more than two trackers. On the other hand, 30 apps had over five trackers. In Table 4.2, the 15 apps that presented the most trackers in our data set are shown, ranging from 11 to 9. The purpose of these apps varies; some have been downloaded hundreds of thousands or even millions of times (it shows “-” if the application is no longer in Google Play).

Given the urgency to provide digital health apps to restrain the spread of the virus at the beginning of the pandemic, it would not be surprising if, initially, applications extensively used third-party libraries and then removed or reduced their use in subsequent releases. Using historical information of application reports from *Exodus*, the number of trackers detailed in each one of these reports was queried to determine whether the tracker number per application decreased over time. The goal was to understand if, since the containment of the virus was successful, governments, organizations, and other providers of COVID-19-related apps improved their privacy aspects.

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 4.1: Number of trackers found in the studied applications, and their respective percentage.

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

TABLE 4.2: Top 15 applications with most trackers included.

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 4.3: Increase and decrease of trackers per application, based on *Exodus* historical reports.

The results can be seen in Table 4.3: for apps that contained more than one privacy report (153 in total), so there could be a comparison, 25 of them ($\approx 16\%$) decreased the count of trackers included, with 6 of these 25 ending with no trackers detected in their latest report; 21 ($\approx 14\%$) increased their count of trackers, with 6 of them starting with no trackers but including at least one at some point; 56 ($\approx 37\%$) maintained the same number of trackers, and 51 apps ($\approx 33\%$) did not include any tracker in any version.

From the perspective of the trackers, 58 different trackers were present in the 595 apps analyzed. In Table 4.4, the results show that trackers detected by *Exodus* such as *Google Firebase Analytics* and *Google CrashLytics* are present in a significant number of the apps of the data set, especially the former (almost two out of three apps contained it). It can also be appreciated that nearly half of the detected trackers, 28 out of 58, were included in no more than two apps. Braze was previously known as Appboy, which appears with this name in the binary packages. Same with ex Urban Airship

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 4.4: Number of applications (out of 595) in which the identified trackers were found, together with their percentage of occurrence.

now known as Airship.

The correlation between the most included trackers was also studied. This can be appreciated in Figure 4.1: 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 4.4: 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.

Application developers are encouraged to add SDKs to their code because of their functionality and services. The extended usage of SDKs is shown in Table 4.5, where it can be seen that almost half the SDKs included were related to analytics (“Perct.” column). Remarkably, five mobile advertisement SDKs were used in our universe of studied apps (“Count” column).

AltBeacon	1	0.083	0	0	0.083	0.083	0	0.083	0.17	0.083	0.67	0.92	0.083	0.083	0	0	0.083	0
Amplitude	0.071	1	0	0.79	0.79	0.79	0.14	0.79	0.79	0.79	0.36	1	0	0.071	0	0	0	0
Bugsnag	0	0	1	0	0.083	0.083	0.083	0.083	0.17	0	0.083	0.25	0	0	0	0	0	0
Fb Ads	0	0.79	0	1	0.93	0.93	0.14	0.93	1	0.79	0.29	1	0	0	0	0	0	0
Fb Analy.	0.029	0.32	0.029	0.38	1	1	0.47	0.97	0.47	0.35	0.41	0.88	0.029	0.029	0	0	0.088	0.029
Fb Login	0.025	0.28	0.025	0.33	0.85	1	0.45	0.95	0.45	0.35	0.45	0.9	0.075	0.025	0	0.025	0.075	0.05
Fb Places	0	0.11	0.056	0.11	0.89	1	1	1	0.28	0.17	0.61	0.83	0.11	0.056	0	0.056	0.17	0.056
Fb Share	0.026	0.29	0.026	0.34	0.87	1	0.47	1	0.47	0.37	0.45	0.89	0.079	0.026	0	0.026	0.079	0.026
Ggle AdMob	0.043	0.23	0.043	0.3	0.34	0.38	0.11	0.38	1	0.36	0.26	0.94	0.085	0.043	0.043	0.043	0.043	0.043
Ggle Analy.	0.036	0.39	0	0.39	0.43	0.5	0.11	0.5	0.61	1	0.54	1	0.46	0.071	0	0	0.036	0.071
Ggle Crash.	0.044	0.028	0.0056	0.022	0.078	0.1	0.061	0.094	0.067	0.083	1	0.99	0.061	0.094	0.0056	0.011	0.05	0.061
Ggle Fireb.	0.031	0.039	0.0084	0.039	0.084	0.1	0.042	0.095	0.12	0.078	0.5	1	0.039	0.056	0.022	0.028	0.053	0.036
Ggle Tag M.	0.071	0	0	0	0.071	0.21	0.14	0.21	0.29	0.93	0.79	1	1	0.071	0	0.071	0	0.14
HMS Core	0.042	0.042	0	0	0.042	0.042	0.042	0.042	0.083	0.083	0.71	0.83	0.042	1	0	0	0.46	0
Ms Analy.	0	0	0	0	0	0	0	0	0.1	0	0.05	0.4	0	0	1	0.95	0.05	0
Ms Crash.	0	0	0	0	0	0.045	0.045	0.045	0.091	0	0.091	0.45	0.045	0	0.86	1	0.045	0
OneSignal	0.042	0	0	0	0.12	0.12	0.12	0.12	0.083	0.042	0.38	0.79	0	0.46	0.042	0.042	1	0
OpenTelem.	0	0	0	0	0.077	0.15	0.077	0.077	0.15	0.15	0.85	1	0.15	0	0	0	0	1
	AltBeacon	Amplitude	Bugsnag	Fb Ads	Fb Analy.	Fb Login	Fb Places	Fb Share	Ggle AdMob	Ggle Analy.	Ggle Crash.	Ggle Fireb.	Ggle Tag M.	HMS Core	Ms Analy.	Ms Crash.	OneSignal	OpenTelem.

FIGURE 4.1: Proportion of concurrent usage of trackers.

Finally, in Table 4.6 we show the number of times trackers in the categories described above were included in the studied apps. While it can be seen that these libraries were broadly used to assist in the development and maintenance of apps in production, trackers offering advertisement services, whose pertinence with the studied apps is debatable, were included 77 times.

4.2 Trackers in Action: Observed Methodology of Trackers

The analyzed trackers indicated some common patterns with their methodology of data harvesting. In this section, the most common methodologies are discussed, based on how they collect the data, when they collect it, and how they send it to their servers. Important to note is that the different perspectives discussed are not mutually exclusive, thus, a tracker can collect the information using a real-time approach, triggered after an event, in a centralized form. For reference, we provide the name of a tracker using each methodology described.

4.2.1 Real-Time vs Store & Retrieve:

- **Real-Time Collection (e.g. Branch)** Trackers using this methodology will harvest all the information on the go. They will start collecting the information in a specific method from a class, and then it will make a chain of methods calls until it reaches the last method, where the data is finally sent to their servers in a POST request.

- **Store and Retrieve Collection (e.g. Amplitude)** In this methodology, trackers first collect the information and immediately store it in classes' fields (AdColony) or in internal databases (MixPanel). At a later moment, this previously stored data is queried by the tracker and sent to its servers using POST requests.

Service	Description	Count	Perct.
App Usage, Audience and Engagement	Analytics of how the app is used, and by who, events monitoring and tracking, user engagement	26	44.83%
Crash Reporting and Monitoring	Monitoring of resources, logs, crashes, and general app functioning	10	17.24%
Technical Functionality	Specific functionality for the user that is integrated into the app (e.g. maps, beacons and code scanning)	8	13.79%
Mobile Advertising	Distribution of ads into the app, monetization	5	8.62%
Devel. and Back-end Framework	Libraries for ease of app development and back-end services (e.g., hosting, storage and distribution)	3	5.17%
Push Notification	Push notification and other messaging channels	3	5.17%
Social Media Integration	Integration with social media within the app (e.g login and content sharing)	3	5.17%

TABLE 4.5: Types of services offered to the developers by the trackers analyzed. Their main service was chosen for trackers that fit more than one category.

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 4.6: For each service category, count of times trackers belonging to these categories were included.

4.2.2 Event-Based vs General:

- **Event-Based Collection (e.g. MapBox)** Trackers using this approach will harvest the data when determined events fire. This data includes information related to the event and any other piece of information that the tracker or app developer wants to collect.

- **General Collection (e.g. Matomo)** Under this approach, trackers collect general information regarding the device, the user, the app and its use, without the need for waiting specific events in the app.

4.2.3 Centralized vs Iterative vs Decentralized:

- **Centralized Collection (e.g. OneSignal)** Here, all the information is collected from a unique point in the tracker, in other words, within a single method of a class. Given the advantages of polymorphism in Object Oriented Programming, some trackers call the same method to collect the data, but the data collected varies based on the class where that method is implemented (Braze).

- **Iterative Collection (e.g. AdMob)** Trackers that harvest data in an iterative way, collect the information in several methods that are called in chain, adding to the set of collected data in previous methods of the chain, the data that is harvested in each successive call.

- **Decentralized Collection (e.g. Flurry)** Under this approach, the tracker harvest the data at different places in its code, that are unrelated to each other. For example, in a method, it can collect and send the device information, and in a different class and method harvest and exfiltrate data about specific app events.

4.2.4 Centralized Connections vs Decentralized Connections:

- **Centralized Connections (e.g. Airship)** In this case, although the tracker could collect the information in a centralized or decentralized way, the requests that are sent to its servers in order to exfiltrate data leave from a single class.

- **Decentralized Connections (e.g. Bugsnag)** 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 from to this category.

4.3 Detailed Information of Analyzed Trackers

This section details, for a selected subset of the studied trackers (one for each methodology discussed above), the information that they gather and then they send to their servers, and its associated high-level data flow, taking as reference a loose UML notation. Tones of **red** were used to indicate the process of harvesting data, whereas **blue** was used to highlight the steps involved in the sending of the captured data to the trackers' servers (a dotted **blue** arrow means that several successive classes are called in between the two extremes of the arrow). The names of classes, methods and fields correspond to the naming made by *JADX* while analysing the app, therefore, for the

cases where obfuscation was used by the developers, the names may appear cryptic (e.g. C12345 for a class, or m67890 for a method). In the cases where obfuscation was not used, the names tend to hint what these classes or methods do, especially when tracking the information. Methods bordered on **black** mean that data is mainly harvested in there. Moreover, in order to bring context to the reader, a brief introduction of the tracker is provided, in addition to a table describing the information that it harvested (more information regarding all the information gathered by trackers can be found in Section 4.4).

The details of the complete set of analyzed trackers can be found in Appendix B.

4.3.1 Airship (Centralized Connection)

Introduction

Formerly known as Urban Airship, it was founded in 2009. Its main goal is to provide marketing and branding services, allowing companies to send custom messages to consumers via push notifications, email, web notifications, SMS messages, among others, enabling customer engagement. It also offers analytics services. Its website is <https://www.airship.com/>.

Tracked Information

Table 4.7 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	Yes
AAID	-
User ID	Session id Push id
Location SW	Timezone Locale country Language Locale variant Daylight saving
Location HW	Latitude Longitude Accuracy Location provider
Device SW	OS version
Device HW	Device manufacturer Device model Device type / Platform / family ("android", "amazon")
APK	App version Package name Package version Production (True, False) Google Play referrer
Applications and Processes	Foreground
Disk and Memory	-
Network	Carrier Connection type Connection subtype
Screen and Audio	Screen Previous Screen
Rooted, Jailbroken, Emulated and Simulated	-
Time	Timestamp (event) Entered time Exited time Duration
Battery	-
SDK	API version SDK/Lib version
Others	Event type Action (enter, exit) Metadata Update Dist

TABLE 4.7: Tracked information by Airship.

Data Flow Diagram

Airship collects the information from three different classes. When it is harvested in *AirshipChannel*, then it is sent to *ChannelAPIClient*, and after that a request is sent to the server with the data as payload. When the data is gathered in *RemoteDataAPIClient*, immediately the *Request* class is called and the data is sent. Finally, when the harvesting happens in *Analytics*, the data collected there is related to an event that was triggered in the app. This data is complemented with further data from a local database (DBO), and then it is sent to the server as a request with a payload as well.

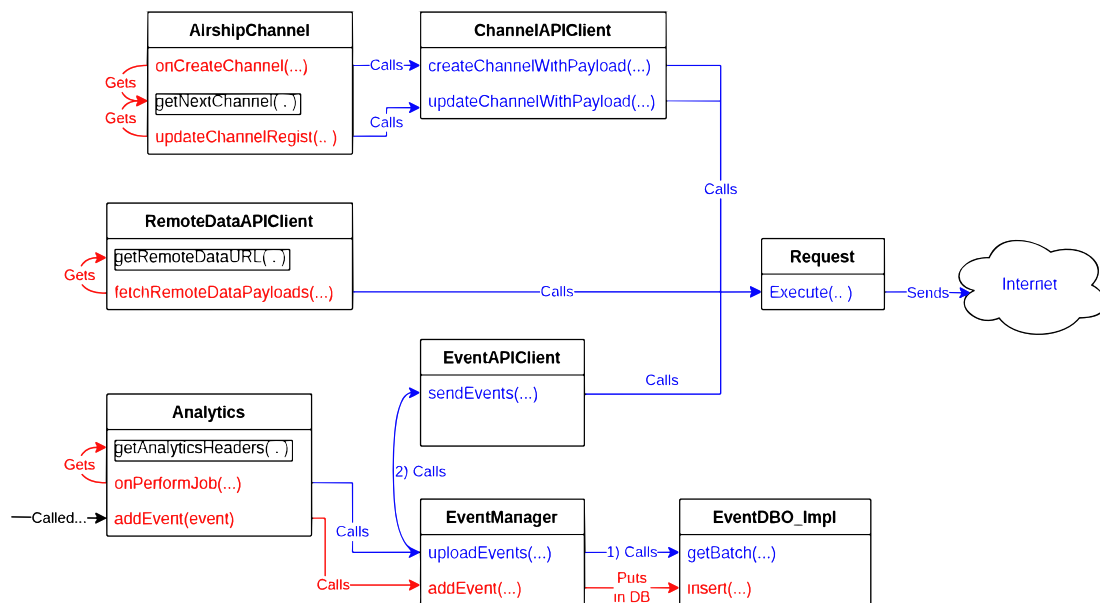


FIGURE 4.2: Data flow in Airship

4.3.2 Amplitude (Store and Retrieve Collection)

Introduction

Founded in 2014, Amplitude provides a series of analytics to help app developers understand how users use their apps. It is one of the most successful trackers and a leader in the analytics market. Its website is <https://amplitude.com/>.

Tracked Information

Table 4.8 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Technical Functionality
Push Notifications	-
AAID	Android id Limited ad tracking
User ID	UUID
Location SW	Country code Language Locale country
Location HW	Latitude Longitude Network Country ISO
Device SW	OS version OS name
Device HW	Device manufacturer Device model Device brand
APK	App version Google Play Services enabled
Applications and Processes	-
Disk and Memory	-
Network	Carrier
Screen and Audio	-
Rooted, Jailbroken, Emulated and Simulated	-
Time	Timestamp (event)
Battery	-
SDK	-
Others	Event type Event sequence number Tracking Options

TABLE 4.8: Tracked information by Amplitude.

Data Flow Diagram

Amplitude's data flow is quite simple. First, it collects the device information each time an event is triggered and stores it in a local database. Later, when the `updateServer(...)` method is fired, it gets all pending events from the local database and sends them to their server as a POST request.

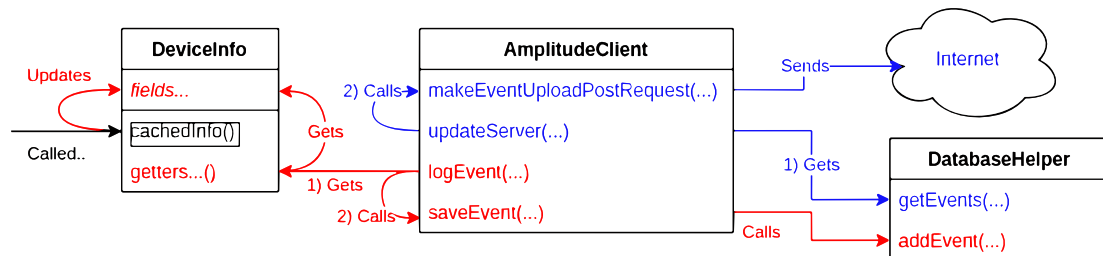


FIGURE 4.3: Data flow in Amplitude

4.3.3 Branch (Real-Time Collection)

Introduction

Another provider of app usage, audience and engagement, Branch was founded in 2014. Also known as Branch Metrics, it helps companies drive seamless mobile experiences through its linking infrastructure. Its website is <https://branch.io/>.

Interestingly, its web-tracking services contained a XSS vulnerability that was publicly reported and had the potential to impact million of users of top-level websites and apps (www.vpnmentor.com/blog/dom-xss-bug-affecting-tinder-shopify-yelp).

Tracked Information

Table 4.9 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	-
AAID	Ad ID
User ID	Unique id
Location SW	Language Country code
Location HW	-
Device SW	User agent (webSettings) OS version OS ("android")
Device HW	Device manufacturer Device model
APK	App version
Applications and Processes	-
Disk and Memory	-
Network	Local IPs WiFi connected
Screen and Audio	UI mode DPI Height Width
Rooted, Jailbroken, Emulated and Simulated	-
Time	First time install Last update time
Battery	-
SDK	SDK ("android") SDK version
Others	Device fingerprint ID identity

TABLE 4.9: Tracked information by Branch.

Data Flow Diagram

Branch's data collection is straightforward. If a Restful GET is made to the server, the query parameters are harvested in class *ServerRequest*. On the other hand, if a Restful POST is made, the device information is gathered in class *DeviceInfo*, which name is self explanatory.

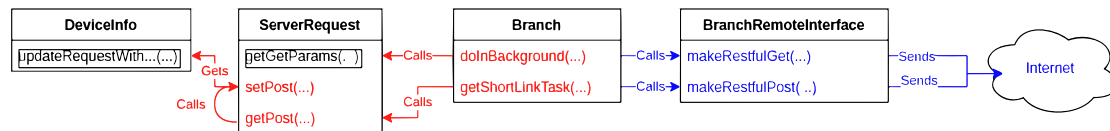


FIGURE 4.4: Data flow in Branch

4.3.4 Bugsnag (Decentralized Connection)

Introduction

Differently to previous discussed trackers, Bugsnag is used for crash reporting and app monitoring, instead of for audience reach, engagement, and monetization. It is owned by SmartBear company, and it provides an error monitoring and application stability management solution. Its website is www.bugsnag.com/.

Tracked Information

Table 4.10 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Crash Reports and Monitoring
Push Notifications	-
AAID	-
User ID	Device id Build UUID Session UUID User UUID
Location SW	Locale
Location HW	Location status ("allowed")
Device SW	App type OS name OS version OS build Android API level
Device HW	Device model Device manufacturer Device brand
APK	App id / Package name Stage ("production") App version App version code App name
Applications and Processes	Active screen (activity)
Disk and Memory	Memory Usage Low memory Total memory Free disk Free memory
Network	Network access (type)
Screen and Audio	Orientation Density DPI Resolution
Rooted, Jailbroken, Emulated and Simulated	Jailbroken Emulator
Time	Duration Duration foreground Time (event/crash) Started at (session) Sent at (connection header)
Battery	Battery level Charging
SDK	Payload version
Others	Notifier Binary arch Code bundle id Autotracking ("true")

TABLE 4.10: Tracked information by Bugsnag.

Data Flow Diagram

In Bugsnag, class *SessionTracker* is in charge of calling classes *DeviceDataCollect* and *Session*, where data is collected. Later, this class calls to *DefaultDelivery* and the gathered data is sent to the servers. At the same time, class *Client* also collects data and sends it to the servers on its own.

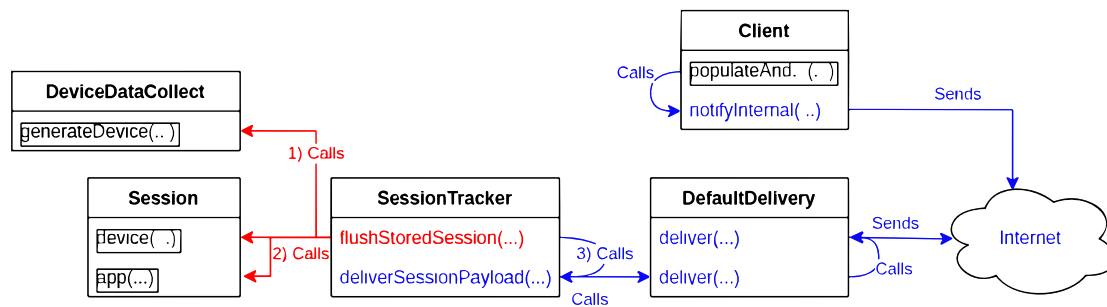


FIGURE 4.5: Data flow in Bugsnag

4.3.5 Flurry (Decentralized Collection)

Introduction

Flurry is another mobile analytics, monetization, and advertising company, which was founded in 2005, therefore is one of the oldest companies studied. It develops and markets a platform for analyzing consumer interactions with mobile applications, packages for marketers to advertise in-apps, as well as a service for applying monetization structures to mobile apps. In 2014 Flurry was purchased by Yahoo!. It can be found at www.flurry.com.

Flurry was notorious in 2010 when Apple changed its ToS after discovering the tracker harvesting device information from their own new devices, as Steve Jobs stated in an interview: “Well we learned this really interesting thing. Some company called Flurry had data on devices that we were using on our campus – new devices. They were getting this info by getting developers to put software in their apps that sent info back to this company! So we went through the roof. It’s violating our privacy policies, and it’s pissing us off! So we said we’re only going to allow analytics that don’t give our device info – only for the purpose of advertising”. (<https://www.engadget.com/2010-06-01-steve-jobs-live-from-d8.html>)

From a privacy perspective, from 2010 to now, things have changed for the worse, and tracking activities have been normalized in the app ecosystem.

Tracked Information

Table 4.11 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Crash Reports and Monitoring
Push Notifications	Yes
AAID	Ad tracking enabled
User ID	Count Device id (advertising id, install id, device id) Crash map id User id Guid (uuid)
Location SW	Locale
Location HW	Latitude Longitude Accuracy
Device SW	OS version release OS architecture Platform (3)
Device HW	Device model Device device Device brand Device board Device id Device product
APK	Package version name Package version code Package name / bundle id
Applications and Processes	State (active, background)
Disk and Memory	JAVA max memory PSS memory RSS memory Free disk Total disk
Network	Net status
Screen and Audio	Orientation
Rooted, Jailbroken, Emulated and Simulated	-
Time	Time Initial run time Timestamp J2
Battery	-
SDK	Agent / SDK version
Others	Number of reports to send Reports content Report type IncludeBackgroundSessionsInMetrics Referrer info Referrer number Launch options Origin Signature keys ETag (5 other elements)

TABLE 4.11: Tracked information by Flurry.

Data Flow Diagram

Flurry collects and sends information from several independent sources. For example, classes *C0713eu*, *C0478ba*, *C07430* and *C0493bj* perform both tasks. In addition, data is also harvested in *C0493bj*, *C07459* and *C0487be*.

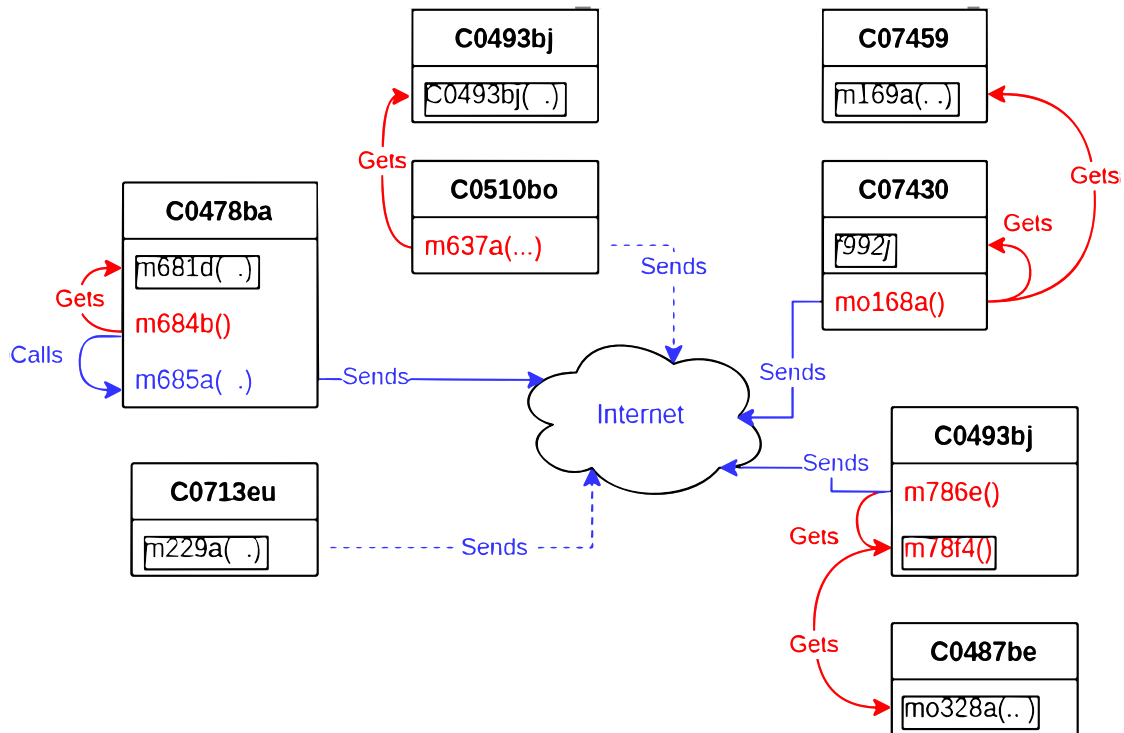


FIGURE 4.6: Data flow in Flurry

4.3.6 Google AdMob (Iterative Collection)

Introduction

AdMob was one of the first mobile advertisement companies, founded in 2006. It was later purchased by Google in 2009. It offers advertising options for several mobile platforms, including Android, iOS, webOS, Flash Lite, Windows Phone and all standard mobile web browsers. Its website is <https://admob.google.com/home/>, under Google's domain.

There was some controversy when Google purchased it. For example, the Center for Digital Democracy and Consumer Watchdog stated that *"The consolidation of AdMob into Google would provide significant amounts of data for targeting advertising"*, and *"The super data profiles that a combined Google/AdMob would facilitate and their use to target advertising raise tremendous privacy issues"* [60].

Tracked Information

Table 4.12 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Mobile Advertising
Push Notifications	-
AAID	Ad id Limited tracking enabled Ad flags Ad format
User ID	Pd id Pd id type
Location SW	Locale Locale country Language
Location HW	Latitude Longitude Accuracy
Device SW	Platform ("Mozilla/5.0 (Linux; U; Android") OS version release OS version SDK User agent (webview) Build fingerprint
Device HW	Device model Device device Device display Device manufacturer
APK	Target API Package version code Package name Is latchesky Is sidewinder Is instant app
Applications and Processes	Activity Is privileged process
Disk and Memory	Runtime mem free Runtime mem max Runtime mem total Private dirty PSS Shared dirty Native private dirty Native PSS Native shared dirty Other private dirty (2 more elements)
Network	Network coarse Network fine Carrier Connectivity network type Telephony network type Phone type Active network type Active network metered
Screen and Audio	Audio mode Music active Speaker on Stream Volume Ringer mode App volume App muted Screen format Density Width pixels (2 more elements)
Rooted, Jailbroken, Emulated and Simulated	Simulator
Time	Time (event)
Battery	Battery level Is charging
SDK	GMB SDK V (3) Is lite SDK SDK environment Lite
Others	Id ("gmob-apps") GWS query id Width height Parent Consent string Consent info Web view count Native version Is non agon (Other fifteen elements)

TABLE 4.12: Tracked information by Google Ad Mob.

Data Flow Diagram

The data collection in Google AdMob involves several clusters of classes and methods calls working independently. Examples are classes *C4855t3* and *C4804o4*. Other examples are classes *C4085rl* and *C3515do*. Finally, in class *el0* data is also harvested, which after several methods called in chain, it is sent to their servers.

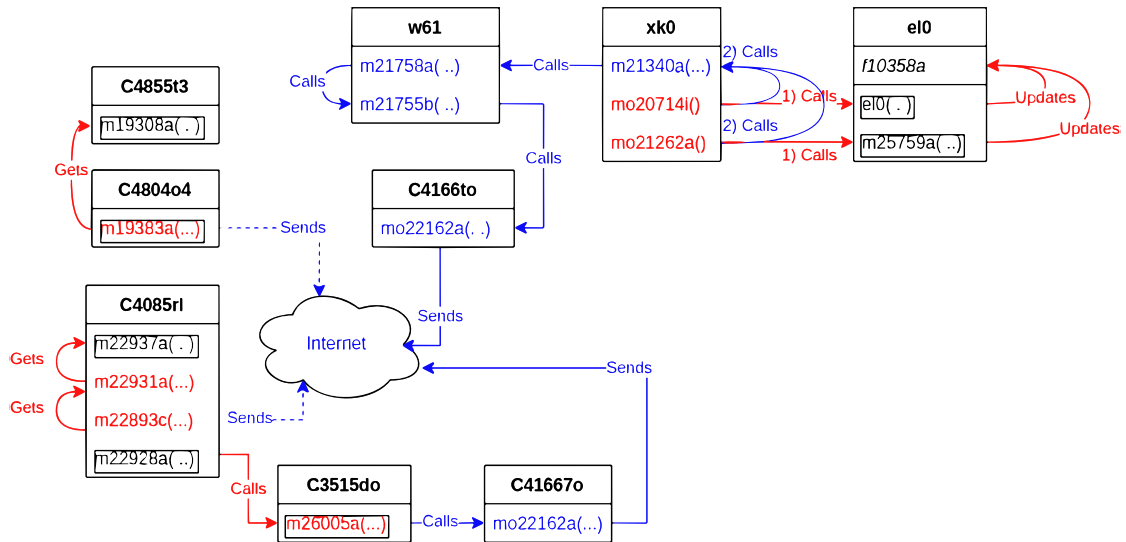


FIGURE 4.7: Data flow in Google AdMob

4.3.7 Mapbox (Event-Based Collection)

Introduction

Mapbox provides integration with online maps for websites and apps, and it is a solid competitor to Google Maps. It was founded in 2012. Its website is www.mapbox.com.

Tracked Information

Table 4.13 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Technical Functionality
Push Notifications	-
AAID	-
User ID	UUID User id Session id
Location SW	-
Location HW	Authorization Horizontal accuracy Altitude Latitude Longitude
Device SW	OS release
Device HW	Device Device model
APK	-
Applications and Processes	App state
Disk and Memory	-
Network	Carrier Cell network type Wifi id
Screen and Audio	Orientation Resolution Font scale
Rooted, Jailbroken, Emulated and Simulated	-
Time	Timestamp Created (event)
Battery	Battery level Plugged in
SDK	Identifier Version SKU id Source ("mapbox")
Others	Event type Enabled telemetry

TABLE 4.13: Tracked information by Mapbox.

Data Flow Diagram

It can be appreciated that Mapbox collects information from two sources. In class *MapboxTelemetry*, information related to events is harvested, which then is sent to the servers. On the other hand, in class *MapboxAccounts* the user ID is gathered and later sent to the servers as well.

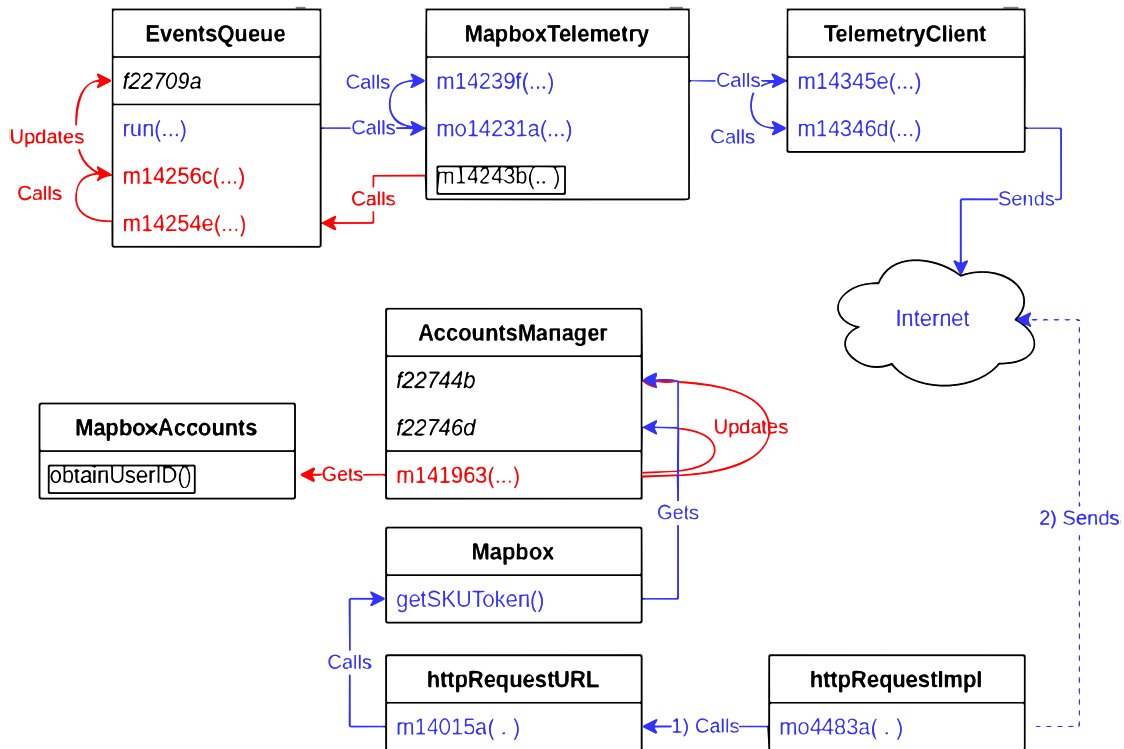


FIGURE 4.8: Data flow in Mapbox

4.3.8 Matomo (General Collection)

Introduction

Previously known as Piwik (founded in 2007, at that time an open-source and free project), currently Matomo provides analytic services and is a direct competitor of Google Analytics. It allows developers to collect data from websites and apps, and then analyze it. Its website is <https://matomo.org/>.

In 2020, a vulnerability with CVSS Score of 10 was found in one of Matomo's docker images provided <https://www.cvedetails.com/cve/CVE-2020-29578/>.

Tracked Information

Table 4.14 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	-
AAID	-
User ID	User id Visitor id
Location SW	Language
Location HW	-
Device SW	-
Device HW	User agent
APK	Package name
Applications and Processes	-
Disk and Memory	-
Network	-
Screen and Audio	Resolution
Rooted, Jailbroken, Emulated and Simulated	-
Time	Timestamp (first event) Previous visit timestamp Date time of request
Battery	-
SDK	Tracker name API version
Others	Opt Out Visit count Session start

TABLE 4.14: Tracked information by Matomo.

Data Flow Diagram

Matomo tracking involves getting information from class *Tracker* (revealing name), and then, altogether data collected from events, it is sent to its servers over the internet.

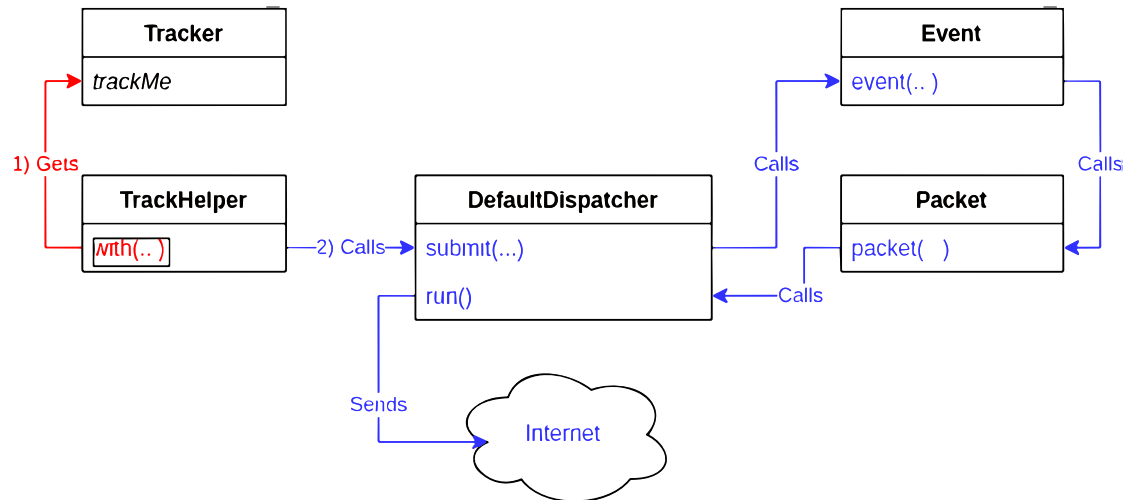


FIGURE 4.9: Data flow in Matomo

4.3.9 OneSignal (Centralized Collection)

Introduction

OneSignal services revolve around customer messaging and engagement, for example, through push notifications, in-app messaging, SMS and email. It was founded in 2014 and its website is located at <https://onesignal.com/>.

Tracked Information

Table 4.15 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Push Notification
Push Notifications	Yes
AAID	Ad id
User ID	Player id
Location SW	Timezone Language
Location HW	Accuracy Time Latitude Longitude
Device SW	Device os Device type
Device HW	Device model
APK	App id Android package App version
Applications and Processes	-
Disk and Memory	-
Network	Network type Carrier
Screen and Audio	-
Rooted, Jailbroken, Emulated and Simulated	Rooted
Time	-
Battery	-
SDK	-
Others	Identifier Subscribable status Events

TABLE 4.15: Tracked information by One Signal.

Data Flow Diagram

OneSignal collects several information in class *OneSignal*, that later is stored in a field of class *UserStats*. As in other trackers, then this field is accessed and its information is finally sent over the internet to the tracker servers (in this case, from class *OneSignalRestClient*).

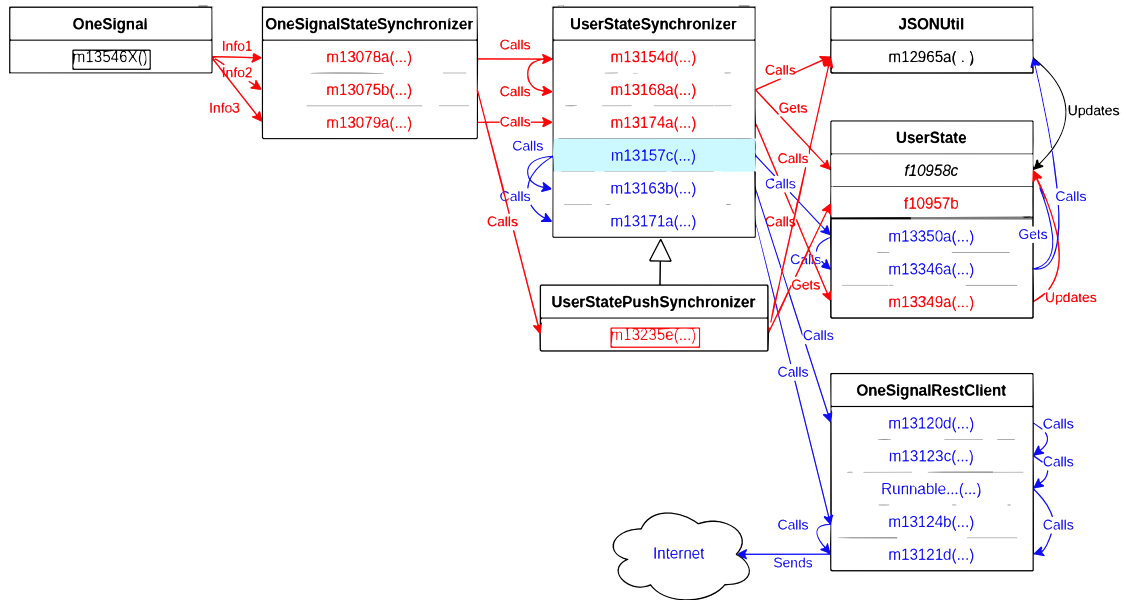


FIGURE 4.10: Data flow in OneSignal

4.4 Tracking the Trackers Tracks: On Data Collection

One of the main findings of this study is that data from the user, device, and usage of COVID-19 apps was being tracked. The methodology of trackers is shown in Figure 4.11. The tracker's company offers mobile developers an SDK that provides certain functionality when integrated into the application (1). Aligned, they setup the SDK server so the tracker can connect back when in use (2). Later, a developer, in this case probably a government, international organization, or someone sponsored by them, will include the SDK into its COVID-19-related application (3), which will be sent to Google Play (4) for release. Mandated by their government, compelled by living circumstances, fearful of being misinformed, or by any other incentive, users will download these COVID-19 apps (5) and start using them (6). While running, at some point, the SDK code will run, including the part dedicated to tracking information, and, at a given time, it will upload the information harvested to the SDK servers (7). Finally, the company behind the SDK will make use of the collected data storing it (8a), for example, to fingerprint the device [37], processing it so they can profile the user of the application (8b) (*Engagement Data* in [20]), or simply sharing it with third parties for a revenue (8c) (point 9 in [83]).

To verify exactly what pieces of information the trackers harvest, the code of the apps shown in Table 4.16 was checked. Since SDKs included in the apps may vary

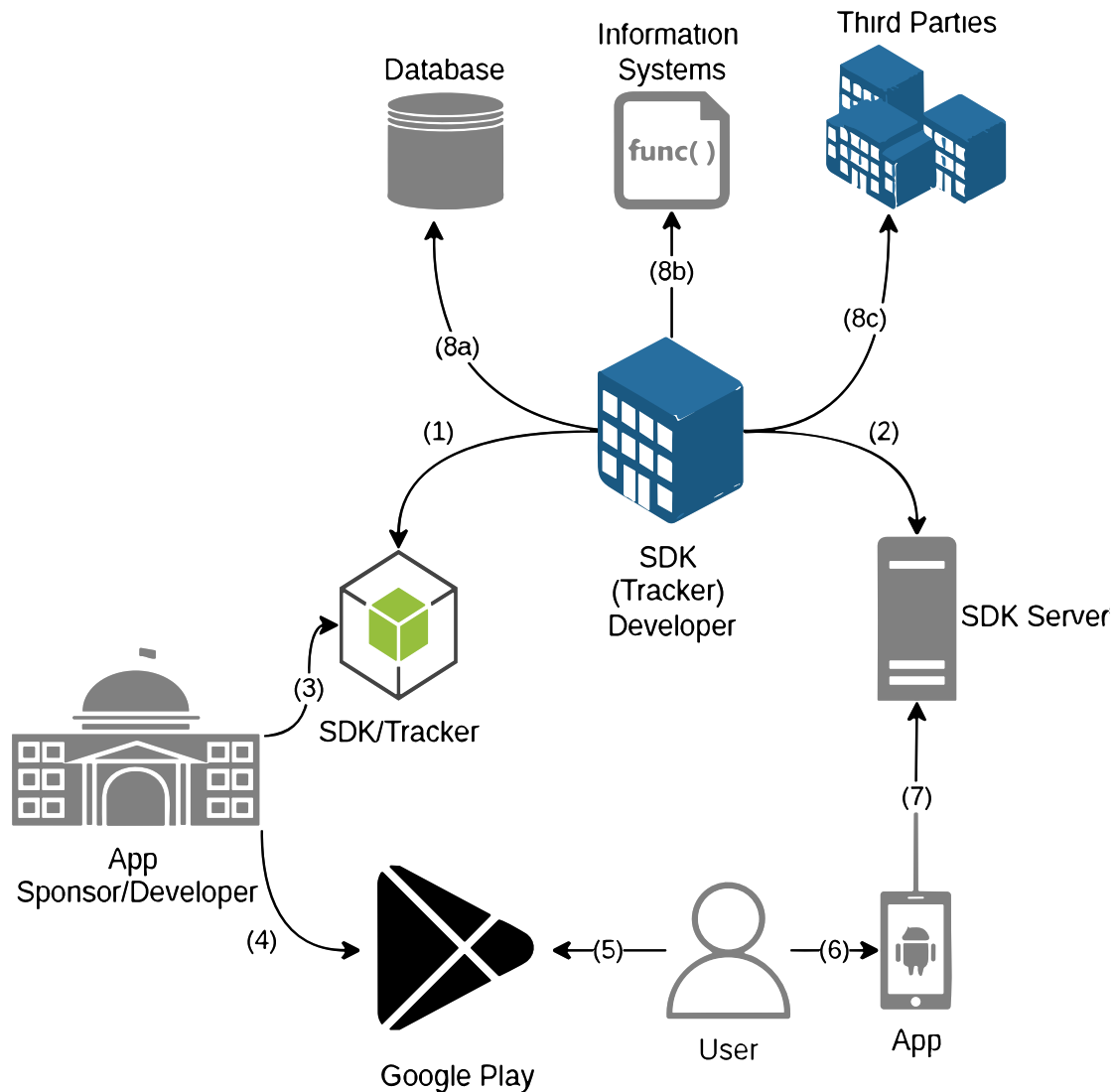


FIGURE 4.11: Tracker data collection example.

between releases, and even the code of the tracker may change over time, their corresponding versions are detailed. For *OpenTelemetry*, two applications were studied to validate our findings. In the case of *Segment*, a pattern in applications developed using the *EXPO* framework (an open-source platform for making universal native apps that run on Android, iOS, and the web) was found, where several wrappers for trackers (including *Segment*) were automatically included in the applications.

Based on the findings of the study, the data gathered by the trackers can be cataloged into the following categories (actual harvesting examples observed in the trackers' code can be found in Appendix C):

Android Advertisement ID (AAID). To this category, as the name suggests, belongs the *Android Advertisement ID* and whether its tracking is limited. This 32-digit string of characters identifier is unique for each Android device, and any application (and any library included in the app) has access to it. Since the same AAID can be accessed by different apps in a device, it allows advertisers build databases profiling customers

Tracker	Tracker Version	Application Package	App. Version
AdColony	4.1.2	covid19.cuernavaca	9.8
Airship	5.1.0	au.com.vodafone.dream-labapp	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.CO-VID19	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.dmrld.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 4.16: Applications that were analyzed to detect tracking behavior.

based on the content they consume and the applications they use.

User ID. Most of the identification artifacts in this category are based on *UUID* variants. Sometimes, the *AAID* was used, or the user ID came directly from the main application code. In a few cases, an ID was assigned at installation time or was set with a push notification. As described above, while the *AAID* is unique for a device and any app that access it gets the same string of characters, the *UUID* scope is for each app and it varies between apps.

Location Software. In this category, it is included tracking information that can approximate the location of the application user, which is based on the software configurations of the device. For example, calls to track the *locale*, *language* and *timezone*, *country code*, *input languages* and *daylight saving*.

Location Hardware. Contrary to the above, the calls to get the user's and device's exact geographic location in this category are grouped here.

Device Software. This category includes the 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. The information about the physical device was grouped within this category, including elements like: the device *manufacturer*, *model* and *brand*, *name*, *id*, *board*, *display* and if it is a *tablet* or *phone*. Also, in this category, the tracking of the *SIM card* is included.

Android Package Kit (APK). Another typical piece of data tracked was the application's information, including the tracker SDK. Elements like the package *name*, *version name* and *version code* belong to this category. Also, information related to the *application bundle*, the *permissions granted*, the *environment*, the development *framework*, if it is an *instaApp* [56], and 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. To this category belong the calls the trackers did to get information about the *carrier*, the *network* used by the device, the *wifi*, and *bluetooth*. For a particular tracker (*Startapp*), the collection of over one hundred different elements related to this category was observed.

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* level, among others.

Rooted/Jailbroken/Emulated/Simulated. A few trackers tried to identify if the device was *rooted/jailbroken* or if it was an *emulation/simulation*.

Time. In this category, time-related tracking activities are grouped. For example, the time of the *application install*, its *last update*, or other *events* that the trackers monitor. An example that is not a timestamp is the *session duration*.

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 Ad-Mob	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Google Fire-base Analytics		✓			✓	✓	✓								✓	✓
Google Tag Manager																
Mapbox		✓		✓	✓	✓		✓		✓	✓		✓	✓	✓	✓
Matomo		✓	✓			✓	✓				✓		✓		✓	✓
MixPanel					✓	✓	✓			✓	✓				✓	
New Relic		✓	✓		✓	✓	✓	✓	✓	✓	✓				✓	✓
OneSignal	✓	✓	✓	✓	✓	✓	✓			✓		✓				✓
OpenTelemetry																
Pushwoosh		✓	✓		✓	✓	✓			✓		✓	✓		✓	✓
Segment	✓	✓	✓		✓	✓	✓			✓	✓				✓	✓
Splunk MINT		✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Startapp	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE 4.17: Type of information harvested by the trackers and sent to their servers.

Battery. Information about the battery was also tracked, for example, its *level* and if it is being *charged*.

Software Development Kit (SDK). The tracker tracked data of itself as well, for example, its *version* and *flavor* (hardcoding these values in their code).

Others. In addition to the categories described above, trackers harvested many other dissimilar types of information from the device and its usage; a clear example of this is the *metadata* and *type* associated with the events they monitored.

In Table 4.17, we detail what information was found to be harvested by each tracker in our analysis of their code. The data tracked is classified based on the categories previously described. *AltBeacon*, *Google Tag Manager* and *OpenTelemetry* are highlighted in green since we could not find tracking behavior in their code for the analyzed version of the SDK. On the other hand, *Google AdMob* and *Startapp* are painted in a red tone given that they track information in every described category.

4.5 Leaking Data in Every Message: On Push Notifications

Another problem found was the use of push notifications to communicate sensitive information to the application users. Indeed, given the type of applications studied, push notifications can include sensitive information, such as exposure alerts, COVID-19 test results, and health-related notifications, among others. Similarly to the data collection functionality, this behavior is illustrated in Figure 4.12. First, the SDK developer will provide its push notification SDK (1) and its associated server (2). As in the previous case, the developer will include the push notification SDK in the application (3) and publish it in the store (4). The user will then download (5) and use the application (6). At the moment of sending a new push notification to all users, a specific group, or a

targeted user of the application; the developers of the apps need to choose the recipients and specify the content of the message to be sent using the service platform at the push notification server (7). This means the message is passed in plaintext to the SDK provider, who can store, process, or transfer this information to a third party (8).

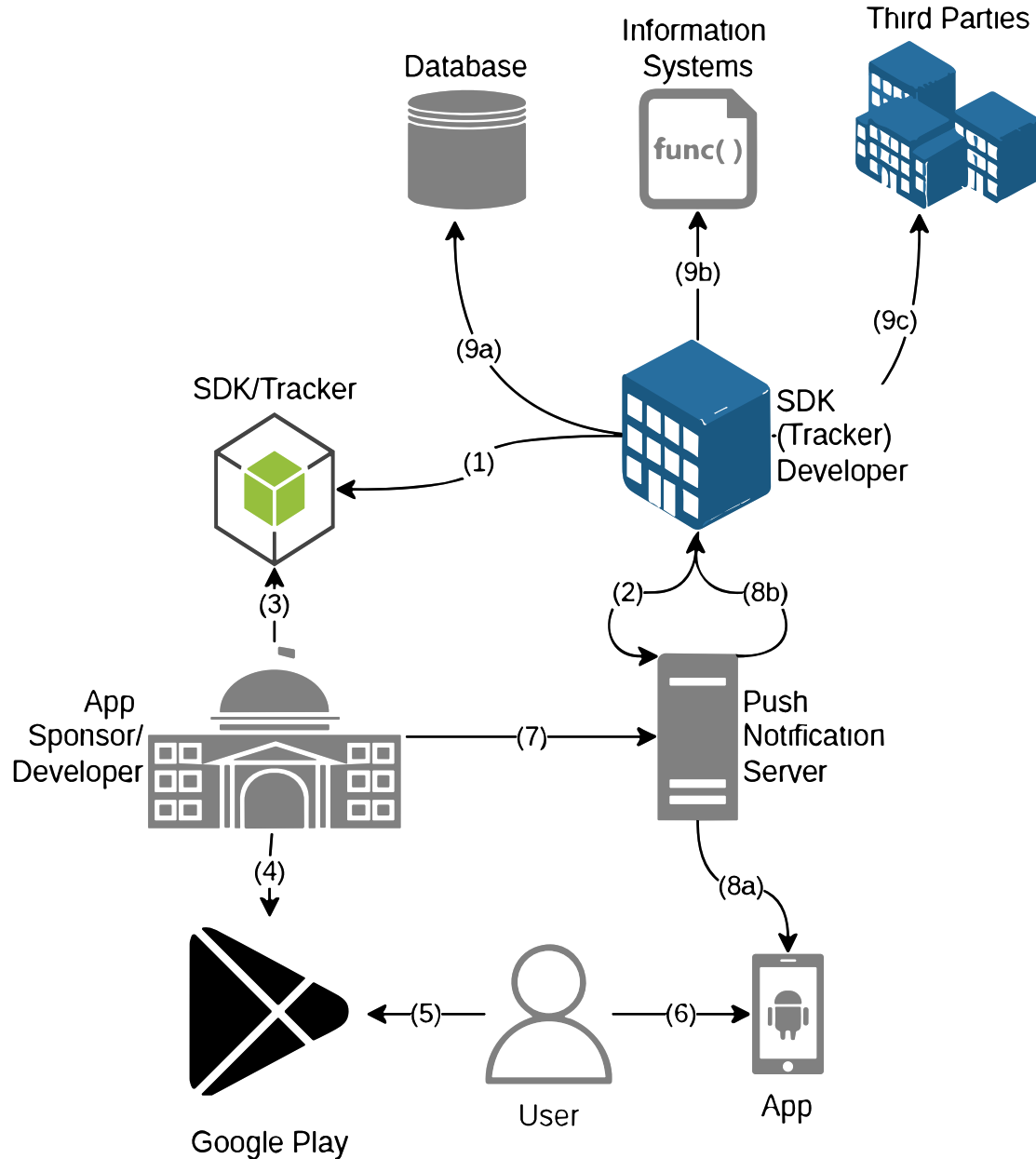


FIGURE 4.12: Tracker push notification example

Out of the 22 trackers analyzed, 6 of them (*Airship*, *Braze*, *Google Firebase Analytics*, *Flurry*, *OneSignal* and *Pushwoosh*) included push notification services to the apps. Reference to end-to-end encryption could not be found in the services documentation, except for the case of *Pushwoosh* [77]. This shall be discussed in Chapter 6.

Other trackers included in the our applications data set that also provide push notification services were: *Adobe Experience Cloud*, *AppMetrica*, *Countly*, *Facebook Analytics*, *HMS Core*, *LotaData*, *MOCA*, and *Pusher*.

Chapter 5

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

This chapter presents the tool we developed, SAPITO, and provides detailed insight about its features.

5.1 Analysing Apps One Hop at a Time: Presenting SAPITO

5.1.1 Motivation for Developing the Tool

One of the biggest challenges faced during this investigation was identifying suspicious packages and the classes and methods of data harvesting. Manually identifying dangerous behaviors of the SDKs is a very time-consuming task. In addition, while *Exodus* is a great tool that instantly flags trackers inside an application, it is based on a database of tracker signatures. If a package signature matches one in the *Exodus* database, that package is flagged as a tracker [46]. This process limits the potential of finding new trackers in the wild that were not yet included in the *Exodus* database. Additionally, *Exodus* does not explicitly detail the data that the trackers harvest.

As a further contribution to this work, a prototype of a tool called SAPITO (SDK Audit and Privacy Investigation TOol) was developed. The purpose of SAPITO is to flag packages included in Android apps as suspicious of leaking sensitive information concerning the phone, the user, and the app usage.

5.1.2 Technology Behind SAPITO

The detection of dangerous SDKs is automatically done through a static analysis of the APK file, which is performed with the assistance of *Androguard* [15]. 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, hence, they can be easily updated (more details in following section). Moreover, it uses Flask for its web interface. Figure 5.1 displays the architecture of the tool.

The tool can be manually accessed via its web interface, or loaded as a separate module in Python 3 to automate privacy checks.

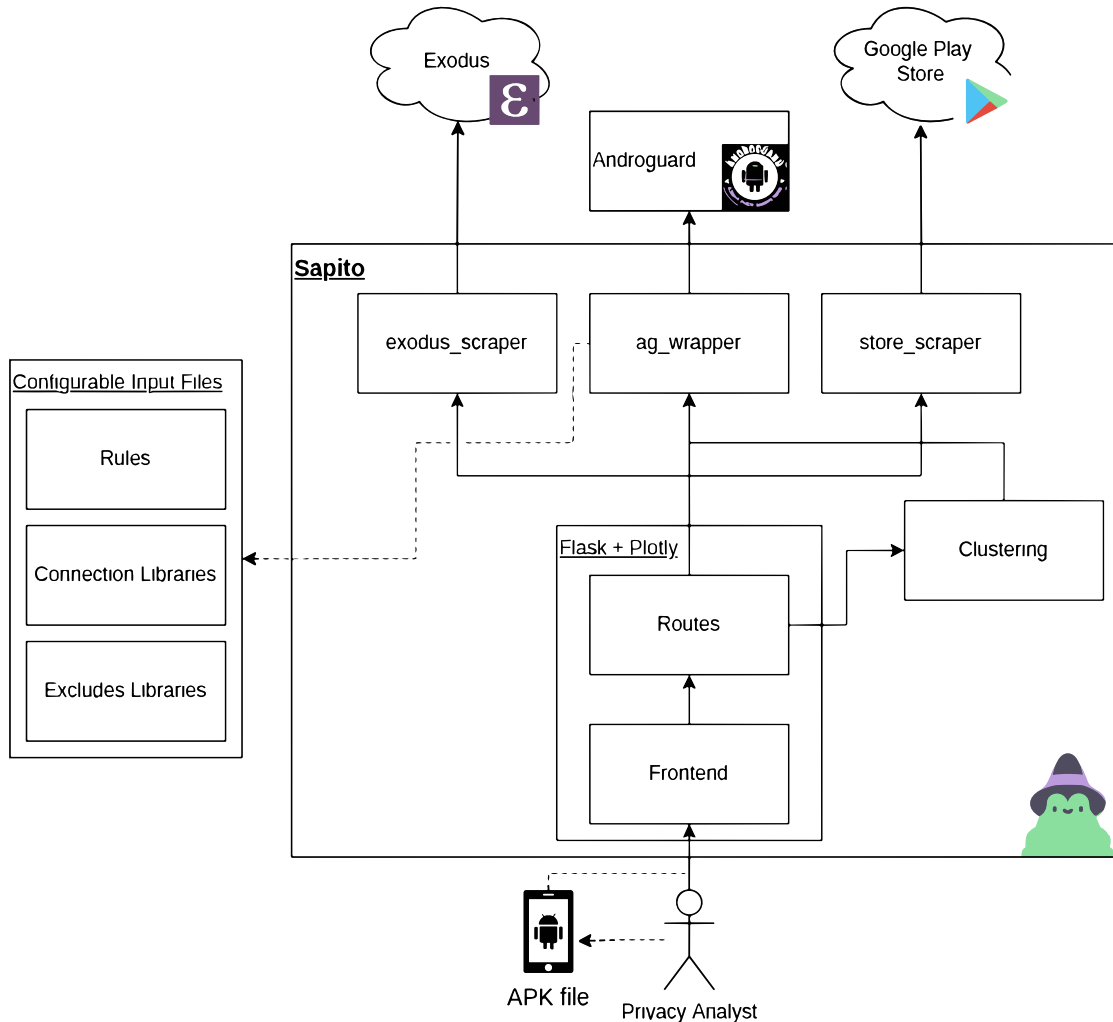


FIGURE 5.1: SAPITO's architecture

5.1.3 What is SAPITO Capable of?

As previously mentioned, SAPITO automatically flags suspicious third-party libraries from Android apps. In order to do so, the privacy analysts, developers, researchers, students, or any individual using SAPITO must start the tool locally in their computer. After the tool is ready for use, the user can access SAPITO's landing page at <http://localhost:5000>.

The landing page is used to load the binaries of the app to be analyzed, hence, a panel is displayed where the user can select the *APK* file of the app from the computer drive.

After loading, processing and analyzing the app, the tool automatically provides a report with packages included in the application showing suspicious activities that were already observed (when doing our manual analysis) on trackers. These are activities such as performing specific cross-references¹, checking for granted permissions to the application in order to piggyback these², monitoring whether the device was rooted/jailbroken/emulated, making use of reflection calls, connecting to the internet, and providing push-notification services. These highlighted libraries would be excellent candidates for further analysis.

The tool also allows the user to analyze in detail these dangerous activities in each third-party library included in the app. For example, a developer may want to know what kind of activities an included library performs in the background when the app is running, and a privacy analyst would like to understand exactly which sensitive data is harvested by the libraries reported. Thus, SAPITO includes several specific analyses.

An option that SAPITO provides is to verify which cross-references each library makes. The goal of this feature is to verify that a library does not call dangerous native methods or fields to harvest sensitive data. Or at least, if these calls are made, they are justified and approved by the developer or privacy analyst. For a better user experience, SAPITO already highlights in red potential dangerous calls, and in yellow suspicious cross-references. We consider dangerous calls any cross-reference to a function/field that was seen used by trackers to harvest sensitive data in our manual study of trackers. On the other hand, suspicious calls are cross-references to classes where dangerous calls are included, but when a dangerous method/field is not exactly called.

Another analysis provided by the tool covers the permissions checks that the selected library makes. This option allows the user to verify that the library is only using appropriate permissions for its correct functioning. For example, a push-notification library may check at run-time if the app has been granted with location access permissions. In that case, it could indicate a malicious behavior by the library that piggybacks on these permissions to harvest location data of the app users, given that, in general, push-notification libraries do not require these kind of permissions to operate.

SAPITO also allows the user to investigate if the library performs checks in order to detect if the device has been rooted or jailbroken, or if it is being emulated or simulated. Most of these checks done by the trackers involve verifying if the app can run the *su* command and if it has access to restricted directories.

Additionally, SAPITO can detect when libraries make reflection calls and, in some cases, even the method called and its parameters. Since reflection can be used to access sensitive information in a rogue way, this particular study may be useful for a privacy or cybersecurity analysis.

¹Also called "XREFs", cross-references are references, calls or invocations to a certain instruction in a program, that could be a function or a data-section (e.g. variable) address for example. In this case, we use it to detect the invocation of functions in certain libraries that could return sensible information.

²For example, an app may require legitimate access to the phone contacts, however, after approval by the user to this permission, any function within the app could read the contact details, including code inside third-party libraries in the app.

The connection calls made by the third-party library can also be analyzed using SAPITO. For this, the most used connection libraries were added into SAPITO internal knowledge information, and if any of them are detected in the code of the third-party library, they are reported to the user.

The user can also investigate if push notifications are being used by the third-party library. As in Chapter 4 will be discussed, push notifications can be a source of sensitive data leakage if private data is transmitted over there (in plain text, as it is the general case).

Additionally, SAPITO uses unsupervised machine learning algorithms to display package clustering. This analysis groups packages that perform similar cross-references, and as a result, it helps to focus the attention on certain libraries that may behave as trackers in the wild (for example, if these end up inside a cluster with known trackers). The two main concepts behind the model are PCA for feature reduction and KMeans for clustering. From a technical point of view, the model creates a data set with all the cross-references present in the code of third-party libraries. Then, using PCA, the data set is shrunk into two dimensions. With this new bi-dimensional data set, four runs of KMeans are processed, iterating on the cluster number, from two to five. The run with the best performance is selected. With the model output, the clusters are displayed using the Python library *Plottly*, given its good user experience (the user can interact in real time with the graph for detailed information). For visualization enhancement, the markers for libraries already detected as suspicious by SAPITO are differentiated, thus, other markers next to these could mean that these libraries could be harvesting data as well.

Finally, the tool is complemented with input from three main sources: The app manifest, *Google Play Store*, and *Exodus* website. General information from the app is gathered from its manifest. The manifest information section not only provides a general overview of the app, since it also details insight about the permissions the app requests. This is important given that libraries included in the app could make use of these permissions (a practice know as permission piggybacking), that are unknowingly granted by the user. Thus, a malicious library could, in fact, harvest these details and exfiltrate them to its servers. On the other hand, *Google Play* extracted information gives a general overview of the app from the public perspective (assuming the app was made public in the store). For example, the user of SAPITO may want to check the app score and reviews in order to detect flaws in the app (potentially related to cybersecurity or privacy), or directly access the privacy policy of the app (or, at the same time, noting the absence of it). Last, the information extracted from *Exodus* website enumerates, for each version of the app that was analyzed in that platform, the trackers included in the code for that version. It is especially useful since it helps to identify trackers already detected that are present in the application (if the app has been analyzed previously). Additionally, since historical information is provided as reports from versions of the app analyzed over the time, the evolution of trackers inclusion in the app can be studied.

Further explanation of the tool features can be seen in the following section. The code and related files can be found at the group's GitLab [79].

5.2 SAPITO Features in Detail

Next, the features that SAPITO currently offers are detailed.

5.2.1 APK Loading

This is the first screen the user sees. The goal of this page is to choose an APK file stored in the user's device, and load it into SAPITO. After the binary is selected and the user clicks on *Analyze!* button, SAPITO begins doing its static reverse engineering analysis using the Androguard wrapper.

In Figure 5.2, it can be seen how the APK of the Uruguayan *Coronavirus UY* app is being loaded.

After the binary is analyzed, SAPITO will redirect the user to the main report page (as detailed next). After that, the user can choose between several detailed information of the app and the detected third-party libraries, at the left-top panel. Additionally, in the left-bottom panel, the detected third-party libraries are enumerated. For *xRefs*, *Permissions*, *Rooted*, *Reflections*, *Connections*, and *Notifications* information option, at least one package must be selected from there (the ones that present a red alarm before their name, were highlighted by SAPITO as dangerous packages, meaning that they could be trackers).

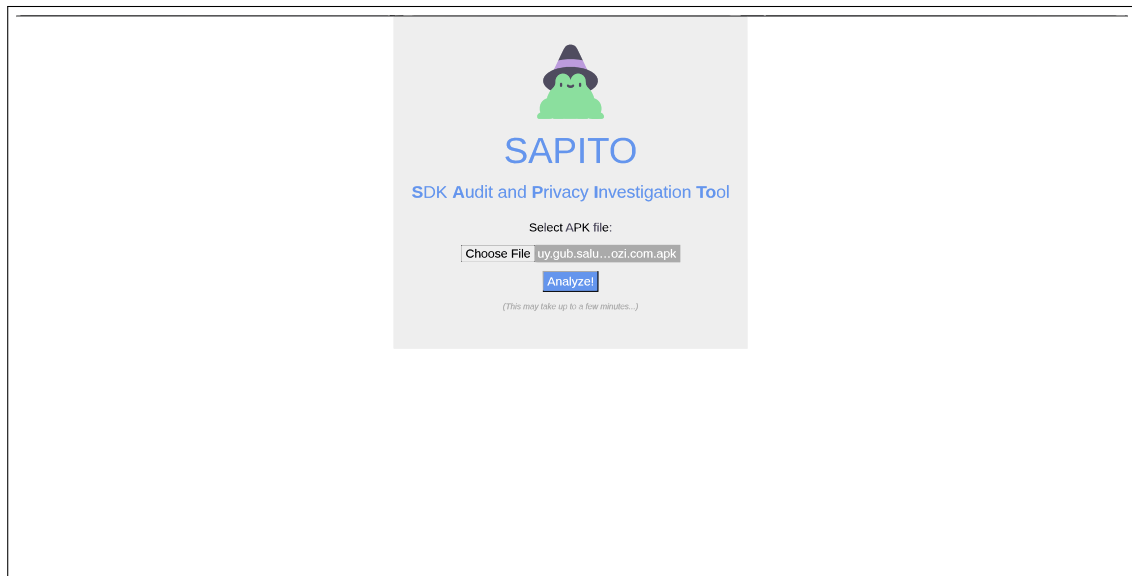


FIGURE 5.2: Landing page in SAPITO, where the user must select a binary to analyze.

5.2.2 Main Report

This is the landing page after SAPITO initial analysis is finished. Suspicious directories are automatically flagged by SAPITO. These directories (as packages proxies) are highlighted since they contain classes that establish internet connections, make calls to get sensitive information about the phone or its usage, checks for permissions or whether the device is rooted, make use of reflection calls, or process notifications. Suspicious colors increase from yellow to red in the badges with the package names, based on the previous detailed dangerous behavior. These third-party libraries can be good candidates for examining, hence, the user can select their check-boxes at the bottom-left panel to get more info about these packages.

In Figure 5.3, it can be seen how the tracker *OneSignal* was flagged by SAPITO (among other potentially dangerous third-party libraries). The report of this tracker indicates it makes internet connections (globe icon), processes notifications (bell icon), checks for permissions (lock icon), uses reflection calls (abc icon), checks if the device is rooted or emulated (danger sign icon), and has dangerous calls within the code (snippet icon) (“dangerous calls” means cross-references that are used to gather sensitive information and that were seen in the manually studied trackers). Since six categories of dangerous behavior are present on this tracker, its name label is in red.

The screenshot shows the SAPITO interface for the analysis of 'Coronavirus UY'. The package name is 'uy.gub.salud.plancovid19uy' and the file is 'uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk'. The report section is titled 'Report' and contains the following text: 'Suspicious directories automatically flagged by Sapito. These directories (as packages proxies) contain classes that establish internet connections, make calls to get sensitive information about the phone or its usage, checks for permissions 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.'

The report lists several packages with their suspicious behaviors indicated by icons (globe, bell, lock, abc, danger sign, snippet):

- com.onesignal** (Red badge): Globe, Bell, Lock, abc, Danger sign, Snippet.
- com.google.android.gms** (Yellow badge): Globe, Bell, Lock, abc, Danger sign, Snippet.
- e** (Yellow badge): Globe, Bell, Lock, abc, Snippet.
- com.genexus** (Yellow badge): Globe, Bell, Lock, abc, Snippet.
- com.google.firebase** (Yellow badge): Globe, Bell, Lock, abc, Snippet.

The left sidebar shows a 'Select packages:' section with a list of packages and checkboxes. The 'com.onesignal' package is checked. The 'Update' button is visible at the bottom of the sidebar.

FIGURE 5.3: Report page in SAPITO, where potential trackers are highlighted.

5.2.3 Packages Clustering

Using PCA for feature reduction and KMeans for clustering, under this option, SAPITO shows how packages may be grouped based on cross-references found in the code of their classes. As the clustering is based on KMeans, different runs of the algorithm will yield different results, though.

In general, a big cluster of close packages will be formed, while some outliers will appear separated from the main cluster. If the privacy analyst already knows 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 (potentially dangerous) cross-references. Packages already detected as suspicious by SAPITO have their markers drawn as 'x' in the graph.

In Figure 5.4, it can be seen how a cluster was formed to the left of the chart, with some other third-party libraries spread to the middle and opposite side of the figure. Since the chart is interactive, pointing with the mouse to any point will yield details of it. In this case, *OneSignal's* point was highlighted, and since it is a known tracker, the three points that appear next to it may be potential trackers as well (One of them was Firebase, which was also analyzed in this thesis, and proved to be a tracker. The other two packages were not analyzed, since they were not flagged by Exodus.).

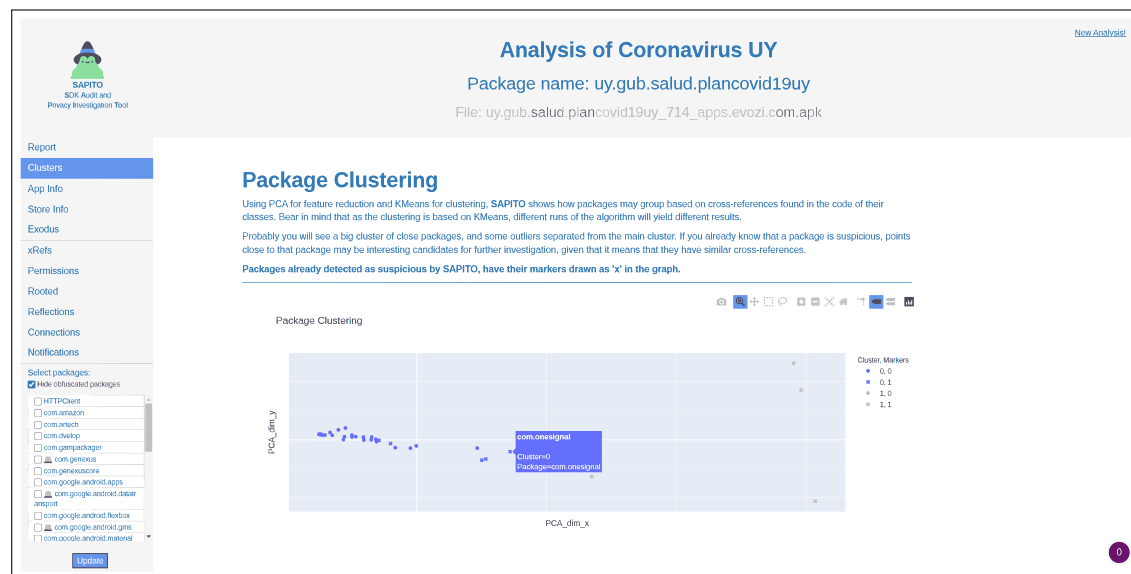


FIGURE 5.4: Cluster page in SAPITO, where trackers are grouped to detect similarities between them.

5.2.4 App Info

This option details general information about the app extracted from its manifest file. Here, while this section is added for the sake of completeness of app information, it may be interesting to check the permissions required by the app, since SDKs may piggyback on these.

The complete information detailed here contains: *App Name, Package, Main Activity, Dex Names, Android Version Code, Android Version Name, Activities, Services, Receivers, Providers, Libraries, Features, Declared Permissions, Permissions, Requested AOSP Permissions Details, Requested Not AOSP Permissions Details, Uses Implied Permissions, Min SDK Version, Max SDK Version, Effective Target SDK Version, and Target SDK Version*. Other information present in the manifest is not included, given its irrelevance.

In Figure 5.5, the first pieces of information from the app's manifest can be seen. Although they do not appear in the figure, in this case, 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*

Dangerous scenarios may appear to the user in case a third-party library within the app would want to use these permissions.

The screenshot shows the SAPITO interface for analyzing the 'Coronavirus UY' app. The page title is 'Analysis of Coronavirus UY' with the package name 'uy.gub.salud.plancovid19uy' and file path 'uy.gub.salud.plancovid19uy_714_apps.evози.com.apk'. The 'App Information' section provides general details extracted from the manifest file, including:

- App Name:** Coronavirus UY
- Package:** uy.gub.salud.plancovid19uy
- Main Activity:** com.anech.activities.StartupActivity
- Dex Names:** classes.dex, classes2.dex

The left sidebar contains navigation options: Report, Clusters, App Info (selected), Store Info, Exodus, xRefs, Permissions, Rooted, Reflections, Connections, Notifications, and a list of packages to analyze, including 'uy.gub.salud.plancovid19uy'.

FIGURE 5.5: App info page in SAPITO, where general information of the app is presented.

5.2.5 Google Play Info

Under this option, general information about the app is detailed, extracted at real time using scraping techniques from its Google Play Store page (if the app is published in the store). As with the previous option, this page is provided so the analyst can have further information regarding the app.

The information provided in here includes: 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 a few users *Comments* left on the store platform.

In Figure 5.6, the first pieces of information from the app's page in Google Play can be seen. The analyst may be interested in checking the app score, or to read a few of the comments, in order to potentially detect issues the app users were facing.

The screenshot shows the SAPITO interface for the 'Store Info' page of the 'Coronavirus UY' app. The page title is 'Analysis of Coronavirus UY' with the package name 'uy.gub.salud.plancovid19uy' and file path 'uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk'. The main content area is titled 'Google Play Store Information' and provides general information about the app. It includes the app's name 'Coronavirus UY', its URL, and a detailed description in Spanish. The description mentions that the app provides medical attention and COVID-19 exposure alerts for residents in Uruguay, allowing users to coordinate tests and receive alerts. It also notes that the app is operated by the Agency of Electronic Government and Society of Information and Knowledge (AGESIC) and the Ministry of Health of Uruguay (MSP). The summary section includes a link to the 'Coronavirus Nacional Plan'.

FIGURE 5.6: Store info page in SAPITO, where information extracted from Google Play is detailed.

5.2.6 Exodus Info

Live information extracted from the Exodus website is presented on this option. Here, the analyst can see the trackers (if any) that were detected by Exodus, if at least one analysis of the app was already performed on that platform. For each analysis for the app found in the Exodus website, here are detailed the app version and the trackers found in that version. In addition, links to the full reports, trackers' information, and trackers' webpage are included.

In Figure 5.7, two of the four total reports can be seen (all the data for version 9.1.0, and just the title for version 7.3.4). It can be seen how the report of the newest version contains 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.

The screenshot shows the SAPITO interface for the analysis of 'Coronavirus UY'. The package name is 'uy.gub.salud.plancovid19uy' and the file is 'uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk'. The 'Exodus Information' section shows live information extracted from the Exodus website. Under 'App version 9.1.0', a table lists the following trackers:

Tracker	Rules	Exodus Info	Tracker Link
Google Crashlytics	io.fabric.com.crashlytics. com.google.firebase.crashlytics com.google.firebase.crash. io.invertase.firebase.crashlytics.	Info	Link
Google Firebase Analytics	com.google.firebase.analytics. com.google.android.gms.measurement. com.google.firebase.firebase_analytics	Info	Link
Huawei Mobile Services (HMS) Core	com.huawei.hms.analytics com.huawei.hms.location com.huawei.hms.plugin.analytics com.huawei.hms.plugin.ads com.huawei.update.sdk com.huawei.agconnect com.huawei.hms.support.tui.push. com.huawei.hms.flutter.analytics.	Info	Link
OneSignal	com.onesignal.	Info	Link

Below the table, the section for 'App version 7.3.4:' is visible but mostly obscured.

FIGURE 5.7: Exodus info page in SAPITO, where privacy reports related to the tracker from Exodus are shown.

5.2.7 Library Cross-References

This section details the cross-references found in the app, in the code of the selected packages. The filter can search for keywords, and the columns header can be clicked to sort the content of the table. Additionally, the content of the table can be downloaded as CSV or JSON.

In Figure 5.8, a few cross-references present in *OneSignal* can be appreciated (since it was the only package selected in the left panel). There can be seen a red (dangerous) and a yellow (suspicious) lines. Cross-references (exact methods or field) that were seen used to gather sensitive data are highlighted in red, while in case when not the exact method or field as seen before were used, but it is another element of a class with a dangerous cross-reference, the call will be appear yellow.

Analysis of Coronavirus UY
 Package name: uy.gub.salud.plancovid19uy
 File: uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk

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/biananz/DeviceMessaging/ADMMessageHandlerBase	<init>	Method	Common
com.onesignal	Landroid/content/Intent	getExtras	Method	Dangerous
com.onesignal	Ljwafang/StringBuilder	<init>	Method	Common
com.onesignal	Ljwafang/StringBuilder	append	Method	Common
com.onesignal	Ljwafang/StringBuilder	toString	Method	Common
com.onesignal	Ljwafang/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	Lohm/ala	completeWakefulIntent	Method	Common
com.onesignal	Landroid/os/AsyncTask	<init>	Method	Common
com.onesignal	Ljwafang/Object	<init>	Method	Suspicious

FIGURE 5.8: xRefs page in SAPITO, where cross-references found in the trackers code are enumerated.

5.2.8 Library Permissions Checks

In this section, the permissions checks performed by the selected trackers can be seen. Also, the usage of these permissions found in the app, in the code of the selected packages, is shown. As before, the filter can be used to search for keywords, and the columns header can be clicked to sort the content of the table. Additionally, the content of the table can be downloaded as *CSV* or *JSON*.

In Figure 5.9, the five times *OneSignal* checks for or use permissions can be appreciated. Of these five, the three permissions are: *RECEIVE_BOOT_COMPLETED* (to start the app as soon as the device finishes booting), *ACCESS_FINE_LOCATION*, and *ACCESS_COARSE_LOCATION* (this and the previous one, to access the location of the device).

Analysis of Coronavirus UY
Package name: uy.gub.salud.plancovid19uy
id.p anco _app. com

Permissions Information
Permissions checks and usage 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.

Search in any column Download CSV | Download JSON

Directory	Permission
com.onesignal	RECEIVE_BOOT_COMPLETED
com.onesignal	ACCESS_FINE_LOCATION
com.onesignal	ACCESS_COARSE_LOCATION
com.onesignal	ACCESS_FINE_LOCATION
com.onesignal	ACCESS_COARSE_LOCATION

Select packages:
 Hide obfuscated packages

- com.google.android.datatransport
- com.google.android.feedback
- com.google.android.gms
- com.google.android.material
- com.google.android.youtube
- com.google.firebase
- com.google.gson
- com.onesignal
- com.plencovid19uy
- com.sinch
- com.squarespace
- com.swirla

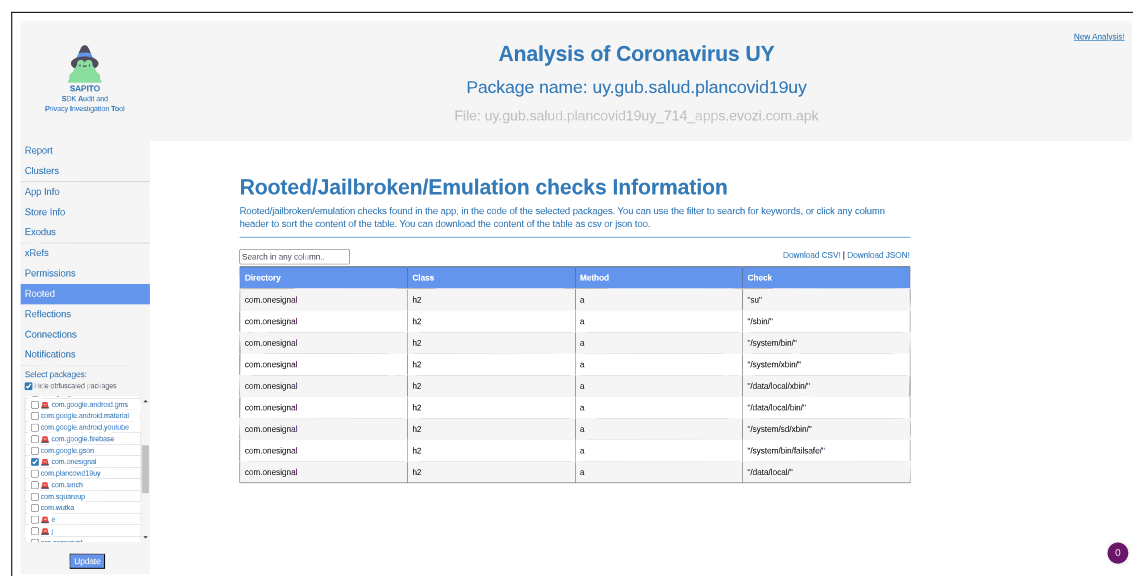
[Update](#)

FIGURE 5.9: Permissions page in SAPITO, where permissions checks found in the trackers code are enumerated.

5.2.9 Library Rooted Checks

Under this option, the checks for the device rooted, jailbroken, emulation or simulation status found in the app by the selected package are detailed. As before, the filter can be used to search for keywords, and the columns header can be clicked to sort the content of the table. Additionally, the content of the table can be downloaded as *CSV* or *JSON*.

In Figure 5.10, these checks made by *OneSignal* can be seen. In this case, and in several others that were found, the tracker would test if it can run the *su* command, and if it has access to several restricted directories, meaning that it has root-level permissions, hence, the device is 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 screenshot shows the SAPITO interface for the analysis of the package *uy.gub.salud.plancovid19uy*. The main content area displays the 'Rooted/Jailbroken/Emulation checks Information' section. Below the title, there is a search bar and links to download the table as CSV or JSON. A table lists the following checks:

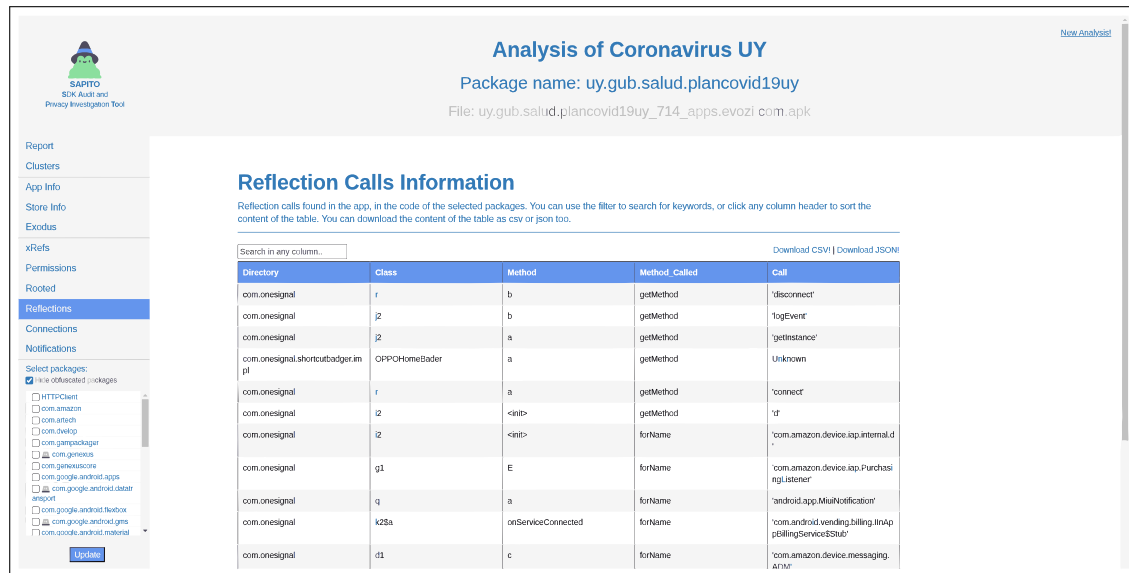
Directory	Class	Method	Check
com.onesignal	h2	a	"su"
com.onesignal	h2	a	"skin"
com.onesignal	h2	a	"system/bin"
com.onesignal	h2	a	"system/lost/bin"
com.onesignal	h2	a	"data/local/bin"
com.onesignal	h2	a	"system/lost/bin"
com.onesignal	h2	a	"system/bin/fixesafe"
com.onesignal	h2	a	"data/local"

FIGURE 5.10: Rooted page in SAPITO, where rooted and similar checks found in the trackers code are enumerated.

5.2.10 Library Reflection Use

This option allows the analyst to identify the reflection calls found in the app, in the code of the selected packages. As before, the filter can be used to search for keywords, and the columns header can be clicked to sort the content of the table. Additionally, the content of the table can be downloaded as CSV or JSON.

In Figure 5.11, the use of reflection made by *OneSignal* can be seen. The importance of analyzing the reflection calls made by a third-party library lies in the fact that these could be considered cross-references that, for some reasons, legitimate or not, are made using this method. A clear example of miss-use of reflection by third-party libraries within Android apps to harvest sensitive data was detected in [89]. In the case of *OneSignal*, while not a dangerous call, it can be seen how at method `<init>` (class constructor) from class `i2`, it calls to method `d` of class `com.amazon.device.iap.internal.d`. Again, obfuscation of code undermines the understanding of these calls.



Analysis of Coronavirus UY
 Package name: uy.gub.salud.plancovid19uy
 File: uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk

Reflection Calls Information
 Reflection calls 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	Method	Method_Called	Call
com.onesignal	r	b	getMethod	'disconnect'
com.onesignal	j2	b	getMethod	'tagEvent'
com.onesignal	j2	a	getMethod	'getInstance'
com.onesignal.shortcutbadger.im	OPPOHomeBader	a	getMethod	Unknown
com.onesignal	r	a	getMethod	'connect'
com.onesignal	i2	<init>	getMethod	'd'
com.onesignal	i2	<init>	fieldName	'com.amazon.device.iap.internal.d
com.onesignal	g1	E	fieldName	'com.amazon.device.app.PurchasingListener'
com.onesignal	q	a	fieldName	'android.app.MiuiNotification'
com.onesignal	k2Ba	onServiceConnected	fieldName	'com.android.vending.billing.InAppBillingService\$Stub'
com.onesignal	d1	c	fieldName	'com.amazon.device.messaging.ARM'

FIGURE 5.11: Reflection page in SAPITO, where reflection calls found in the trackers code are enumerated.

5.2.11 Library Connections

As its name implies, this option enumerates the use of connection libraries found in the app, in the code of the selected packages. To detect these connection calls, the most used connection functions, classes, and third-party libraries for that goal were added to a set, therefore, if the package code makes a call present in that set, it means that it is establishing a connection with a remote server, with the potential of data exfiltration. As before, the filter can be used to search for keywords, and the columns header can be clicked to sort the content of the table. Additionally, the content of the table can be downloaded as *CSV* or *JSON*.

In Figure 5.12, it can be seen how *OneSignal* calls several times to *java/net* connection library, specifically to the class *URLConnection* and its methods for sending data and closing the connections.

Analysis of Coronavirus UY
 Package name: uy.gub.salud.plancovid19uy
 File: uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk

Connection Calls Information
 Use of connection libraries 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 tables as csv or json too.

Search in any column:

Download CSV | Download JSON

Directory	Calling Class	Calling Method	Connection Library	Called Class	Called Method
com.onesignal	r1	d	java/net	URLConnection	disconnect
com.onesignal	r1	d	java/net	URLConnection	getOutputStream
com.onesignal	r1	d	java/net	URLConnection	getInputStream
com.onesignal	r1	d	java/net	URLConnection	setDoOutput
com.onesignal	r1	d	java/net	URLConnection	setDoInput
com.onesignal	r1	d	java/net	URLConnection	setRequestMethod
com.onesignal	r1	d	java/net	URLConnection	setConnectTimeout
com.onesignal	r1	d	java/net	URLConnection	getInputStream
com.onesignal	r1	d	java/net	URLConnection	getResponseCode
com.onesignal	r1	d	java/net	URLConnection	getHeaderField
com.onesignal	r1	d	java/net	URLConnection	setReadTimeout
com.onesignal	r1	d	java/net	URLConnection	setUseCaches
com.onesignal	r1	d	java/net	URLConnection	setRequestProperty

FIGURE 5.12: Connections page in SAPITO, where connections calls found in the trackers code are enumerated.

5.2.12 Library Push Notifications Use

Finally, the last feature of SAPITO (as for now) details the notification service calls found in the app, in the code of the selected packages. As before, the filter can be used to search for keywords, and the columns header can be clicked to sort the content of the table. Additionally, the content of the table can be downloaded as *CSV* or *JSON*. A SDK that provides notification services may be dangerous since the content of these notifications may be processed in plain text by the SDK servers, without the awareness and consent of the app users.

In Figure 5.13, it can be seen how *OneSignal* processes push notifications within its code.

Analysis of Coronavirus UY
 Package name: uy.gub.salud.plancovid19uy
 File: uy.gub.salud.plancovid19uy_714_apps.evozi.com.apk

Notifications Use Information

Notification service calls 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.

SDK providing notification services may be dangerous since their content may be processed in plain text by the SDK servers, without the awareness of the app users.

Directory	Class	Method	Library Called	Method Called
com.onesignal	o1	a	LandroidApp/Notification;	getGroup
com.onesignal	o1	c	LandroidApp/Notification;	getGroup
com.onesignal	o1	a	LandroidApp/Notification;	getGroup
com.onesignal	o1	a	LandroidApp/Notification\$Builder;	recoverBuilder
com.onesignal	o1	a	LandroidApp/Notification\$Builder;	<init>
com.onesignal.shortcutbadger.im	XiaomiHomeBadger	a	LandroidApp/Notification\$Builder;	<init>
com.onesignal	o1	a	LandroidApp/Notification\$Builder;	setSnap
com.onesignal	o1	a	LandroidApp/Notification\$Builder;	build
com.onesignal.shortcutbadger.im	XiaomiHomeBadger	a	LandroidApp/Notification\$Builder;	build
com.onesignal.shortcutbadger.im	XiaomiHomeBadger	a	LandroidApp/Notification\$Builder;	setContentTitle

FIGURE 5.13: Notifications page in SAPITO, where push notification usage found in the trackers code are enumerated.

5.2.13 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, TimeZone, telephony, os, net, etc. This file can be expanded or pruned based on the research needs.

The file *rules.json* that comes with SAPITO contains:

```
{
  "location": ["Accuracy", "Time", "Latitude", "Longitude", "Altitude", "
    Provider", "VerticalAccuracyMeters", "Bearing", "Speed", "CountryCode",
    "AdminArea", "Locality", "ProviderEnabled", "LastKnownLocation", "
    AllProviders", "FromLocation"],
  "Locale": ["Language", "Country", "Variant", "DisplayLanguage", "
    DisplayCountry", "DefaultLocale"],
  "Calendar": ["Timezone"],
  "TimeZone": ["DefaultTimezone"],
  "util": ["Density", "DensityDpi", "WidthPixels", "HeightPixels", "Xdpi", "
    Ydpi"],
```

```

"telephony": ["NetworkOperatorName", "NetworkOperator", "NetworkType", "
NetworkCountryIso", "SimOperator", "SimCountryIso", "PhoneType", "
PhoneCount", "SimState", "SimOperatorName", "DataState", "CellLocation",
"ManufacturerCode", "AllCellInfo", "CallState", "Registered", "
CellConnectionStatus", "CellSignalStrength", "TimingAdvance", "
SubscriberId", "GroupIdLevel1", "DefaultDataSubId", "DefaultSmsSubId", "
DefaultSubId", "DefaultVoiceSubId", "MultiSimConfiguration", "GsmLac", "
GsmCid", "GsmPsc", "CdmaBaseStationId", "CdmaBaseStationLatitude", "
CdmaBaseStationLongitude", "CdmaNetworkId", "CdmaSystemId", "State", "
DuplexMode", "IsManualSelection", "ChannelNumber", "
GsmCellSignalStrength", "GsmRegistered", "GsmTimeStamp", "
GsmCellIdentity", "GsmMcc", "GsmMnc", "GsmCid", "GsmLac", "GsmPsc", "
GsmArfcn", "GsmBsic", "GsmDbm", "SignalStrength", "CellSignalStrengths",
"Level", "CdmaEcio", "LteCqi", "LteRsrp", "LteRssnr", "LteRsrq", "
LteRssi", "LteRegistered", "LteCellSignalStrength", "LteTimeStamp", "
LteCellIdentity", "LteMcc", "LteMnc", "LteCi", "LtePci", "LteTac", "
LteEarfcn", "LteDbm", "LteTimingAdvance", "NrDbm", "NrCsiRsrp", "
NrCsiRsrq", "NrCsiSinr", "NrSsRsrp", "NrSsRsrq", "NrSsSinr", "NrTac", "
NrPci", "NrTimeStamp", "NrNrArfcn", "NrNci", "NrMccString", "NrMncString",
"NrRegistered", "NrTimeStamp", "NrCellIdentity", "LteSignalStrength",
"LteDbm", "Dbm", "GsmEcno", "LteCqi", "LteRsrp", "LteRsrq", "LteRssnr",
"WcdmaRscp", "UsingCarrierAggregation", "DataEnabled", "NetworkRoaming",
"From", "ActiveSubscriptionInfoCount", "ActiveSubscriptionInfoCountMax",
"ActiveSubscriptionInfoList", "CarrierName", "CountryIso", "IccId", "
Mcc", "Mnc", "SimSlotIndex", "SubscriptionId", "DataRoaming", "
WcdmaCellIdentity", "WcdmaRegistered", "WcdmaTimeStamp", "
WcdmaCellSignalStrength", "WcdmaDbm", "WcdmaMcc", "WcdmaMnc", "WcdmaPsc",
"WcdmaUarfcn", "WcdmaCid", "WcdmaLac", "CdmaCellIdentity", "
CdmaRegistered", "CdmaTimeStamp", "CdmaSystemId", "CdmaNetworkId", "
CdmaBasestationId", "CdmaLatitude", "CdmaLongitude", "CdmaDbm", "
CdmaCdmaDbm", "CdmaEvdoDbm", "CdmaEvdoEcio", "CdmaEvdoSnr"],
"net": ["ActiveNetworkInfo", "NetworkCapabilities", "ActiveNetworkMetered",
"ActiveNetwork", "Transport", "ConnectionInfo", "WifiEnabled", "
ScanResults", "CalculateSignalLevel", "ScanSSID", "ScanBSSID", "Level",
"Timestamp", "NetworkId", "WifiSSID", "Type", "TypeName", "Subtype", "
SubtypeName", "Connected", "State", "DetailedState", "Roaming", "
MobileRxBytes", "MobileTxBytes", "TotalRxBytes", "TotalTxBytes", "
WifiRxBytes", "WifiTxBytes", "MobileIfaces"],
"InetAddress" : ["HostAddress"],
"rooted": ["Rooted", "Jailbroken", "Simulated"],

```



```

"os": ["Version.Release", "Version.Incremental", "Version.SDK", "Model", "
Brand", "Manufacturer", "Id", "Supported_Abis", "Display", "Fingerprint"
, "Device", "Board", "Product", "RootDirectory", "
ExternalStorageDirectory", "DataDirectory", "AvailableBlocksLong", "
BlockSizeLong", "Init", "BlockSize", "AvailableBlocksLong", "
AvailableBlocks", "BlockCountLong", "BlockCount", "FreeBlocksLong", "
ElapsedRealtime", "UptimeMillis", "TotalPss", "DalvikPrivateDirty", "
DalvikPss", "DalvikSharedDirty", "NativePrivateDirty", "NativePss", "
NativeSharedDirty", "OtherPrivateDirty", "OtherPss", "OtherSharedDirty",
"Pid", "Uid", "ScreenOn", "Interactive", "PowerSaveMode", "
BatteryManager"],
"content": ["ContextPackageName", "SystemService", "Resources", "FilesDir",
"CheckPermission", "Configuration", "Orientation", "FontScale", "
ScreenLayout", "UiMode", "Locale", "Locales", "VersionCode", "
VersionName", "PackageName", "InstallerPackageName", "ApplicationLabel",
"ApplicationInfo", "InstalledApplications", "FirstInstallTime", "
LastUpdateTime", "SystemFeature", "TargetSdkVersion", "MetaData", "Flags
", "RequestedPermissions", "Query", "SystemAvailableFeatures", "
InstalledPackages", "StringExtra", "Action", "Extras", "IntentFilter"],
"app": ["ProcessMemoryInfo", "MemoryClass", "RunningAppProcessInfo", "
ProcessInfoImportance", "ProcessName", "TaskInfo", "IsRunning", "
MyMemoryState", "MemoryInfo", "LowMemory", "Threshold", "AvailMem", "
TotalMem", "RunningTasks", "TopActivity", "BackgroundRestricted", "
RunningAppProcesses", "NotificationsEnabled", "StorageEncryptionStatus",
"CurrentModeType", "NotificationChannelImportance", "Sound", "
QueryUsageStats", "AppInactive", "TotalTimeInForeground"],
"AdvertisingIdClient": ["AdvertisingIdInfo", "LimitAdTrackingEnabled", "Id"
],
"notificationManager": ["NotificationManagerCompat"],
"webkit": ["DefaultUserAgent", "UserAgentString"],
"view": ["Parent", "BoundingRects", "CurrentInputMethodSubtype", "Locale", "
EnabledInputMethodSubtypeList"],
"System": ["Property", "CurrentTimeMillis"],
"Thread": ["CurrentThread", "Id", "Name"],
"permission": ["AndroidPermission"],
"UUID" : ["RandomUUID"],
"Date" : ["Date", "Time"],
"Runtime": ["Runtime", "Exec", "TotalMemory", "MaxMemory", "FreeMemory"],
"File" : ["UsableSpace", "Init"],
"Battery" : ["Battery_Changed"],
"provider": ["SettingsSecure", "SettingsSystem", "SettingsGlobal", "
INTERNAL_CONTENT_URI", "EXTERNAL_CONTENT_URI", "LocationProvidersAllowed
"],
"bluetooth": ["DefaultAdapter", "Enabled", "BondedDevices", "BondState", "
Name", "BluetoothClass", "Address", "BondState", "DeviceClass"],
"media": ["Mode", "SpeakerphoneOn", "StreamVolume", "RingerMode", "
StreamVolume", "WiredHeadsetOn", "Devices", "MusicActive", "Type"],
"Commands": ["Meminfo", "Version", "Uname", "Cat"],

```

```
"instantApp": ["InstantApp"],
"Kotlin": ["KotlinVersion"],
"security": ["CleartextTrafficPermitted"],
"accounts": ["AccountsByType"],
"StackTraceElement": ["ClassName", "MethodName"],
"hardware": ["Name", "Vendor", "Version", "MaximumRange", "Power", "
    Resolution", "accuracy", "timestamp", "values"],
"Class": ["Name", "Method", "DeclaredMethods"],
"Object": ["Class"],
"reflect": ["invoke"],
"others": ["Amazon", "GooglePlayServicesAvailable", "ReactNativeVersion", "
    Audios.raw", "Www.res", "LiteSDK", "ActivityTransitionEvent", "
    MainAction", "Method"]
}
```

Chapter 6

Discussion

In this chapter, the findings of the study are elaborated. We highlight two problems third-party libraries generate: the data collection of trackers, and the disclosure of sensitive data with push notification service providers. Additionally, further details are presented regarding the posture taken by governments and trackers with data privacy. Finally, potential causes for the issue studied in our research are discussed.

6.1 On the Known Practices of Device Identification: Impact of Data Collection

Device fingerprinting and identification with user segmentation and attribution goals is not an unknown practice. There is already academic research in this field of study, for example, proposing efficient fingerprinting mechanisms to improve authentication [93]. Even trackers openly declare in their privacy policies the purpose of data collection “...for device identification and attribution” [20]. This process is even easier if they collect the *ADID* of the phone.

However, in COVID-19-related apps, this tracking and identification behavior can pose a very high risk to users and their health-related sensitive data. For example, 34 of the total apps processed were mainly used for quarantine enforcement. Of these 34, 29 contained trackers that, among other information, collected the APK identifier. Therefore, if companies behind these trackers knew these APKs were used for quarantine enforcement, they could infer that it was highly likely that the device user was infected with COVID-19. From another perspective, governments, through their apps, sent third parties the COVID-19 infection status of their citizens without their knowledge or consent. Moreover, since some trackers collect location information, they could know, with a GPS level of detail, where users of the devices were being infected. The same reasoning could be applied, for example, to apps focused on vaccination scheduling and COVID-19 certification. Also, they could know the places the user visits, potentially identifying the times the user goes to hospitals. Furthermore, since some trackers collect the SSID of WI-FI hotspots at a range, they could validate that the user is at a particular place.

6.2 A Design Issue: Impact of Using Push Notification Services

As stated in Chapter 4, developers need to send the message content to the push notification service provider in plaintext. Hence, messages like “Your test results were positive...” or “You are scheduled for your X COVID-19 vaccination dose...” have the potential to disclose sensitive health information of the users to third parties. Even worse is the fact that, again, the application users are not aware that this is happening, and they did not give consent to this specific information disclosure to third parties.

Even in cases where encryption was used to protect the notification contents’ confidentiality, the practice was arguably problematic. Based on asymmetric cryptography, a key pair can be created on the user’s device, sending the public key to the SDK servers. Any push notification would then be end-to-end encrypted so that only the device can decrypt the message and display it to the user. Therefore, other parties would not be able to read the push notification content as they are delivered to an appropriate device. However, the problem lies in who are the ends of this end-to-end encryption scheme. This approach would preserve privacy if one end were the user and the other the application developer. However, in all studied cases that implement this, the encryption happens at the service provider servers, so they process the message in plaintext until it is encrypted.

What are the alternatives for the developers? In the context of COVID-19 sensitive information being sent over push notifications, they should rethink their needs. Are they strictly mandatory? Switching to alternatives like email or SMS may face the same problem, but a phone call could avoid this problem. Alternatively, the push notification content could be just an alert, indicating the user to open the application for further details. Another option could be the use of real end-to-end encryption libraries (from the developer servers to the device), like *Capillary* [25].

6.3 Data Privacy Posture from Governments

As commented in Chapter 4, while data privacy is of utmost importance (not in vain, privacy is considered a human right [75]), in a pandemic that puts the entire world at risk, understandably it might be regarded by governments and people as a lower priority. Therefore, with the hope the early launching of an app would help contain the pandemic, it could be justified for some trackers to be included within the app’s code. However, what is not a display of data privacy due care from governments and other reputable organizations is that after the first releases of the apps (and with the pandemic under relative control), the number of trackers included in those apps did not decrease. Moreover, it can be seen in Table 4.3 that this number increased over time during the pandemic in some apps.

Furthermore, several application privacy policies were checked for the research. It was found that transparently informing users of the trackers’ presence within the application was not a practice that all governments followed [53, 74, 44]. On the other hand,

there were cases where developers explicitly declared the included third-party libraries, allowing users to be aware of their presence in the apps; for example, “*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*” [35] and “*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.*” [31].

Additionally, some governments have conducted Data Protection Impact Assessments (DPIA) [19, 43, 38] to identify and minimize data protection risks of their COVID-19-related apps. That is a huge step forward from the perspective of the international normalization of efforts to protect data privacy and enforce privacy-by-design methodologies. Concerning the analyzes of trackers inclusion within the application, some state *...there will be no third-party trackers gathering personal data in the app...* [41], or evaluate, for example, the impact of using *Firebase* for push notifications or the SDK of *Microsoft* to diagnose performance and stability issues [42]. Meanwhile, as with privacy policies, some DPIAs omitted the tracker’s impact analyzes [39, 40].

Hence, it could be concluded that in some cases, on the one hand, users did not have complete information on the privacy policies of the apps they were required to download, and, on the other hand, governments took decisions regarding the release of COVID-19 related apps without completely understanding the impacts of trackers inclusion.

6.4 What the Trackers Declare

It was found that, generally, trackers publicly declare what data they collect, explicitly enumerating the collected items in their privacy policies. However, some claim fewer items than were found during the code analyzes of this study. Therefore, since we cannot have a total trust in the third-party libraries ecosystem, detailed privacy analyzes and tools like SAPITO are indispensable, especially in very privacy-sensitive apps.

The overall results can be appreciated in Table 6.1. 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 the findings of our research.

It could be estimated that Google’s requirement for their *Google Play’s new safety section* [50] has encouraged (or, up to a point, mandated) trackers to transparently indicate the data they collect with their SDKs.

While *AdColony* in its privacy policy details some items being collected using the wording “*Our technology Platform collects and uses personal data, including but not limited to:...*”, they missed declaring, for example, that they collect information about the screen

and battery of the device. Something similar happens with *AppNext*, where some elements collected are not mentioned, for example, if the camera was activated in the last 15 minutes, if the device is charging, and information about the screen and audio system, among others. In the case of *Braze*, there are hints it may collect the *Android ID* along with other information. However, this element was not declared in the sources checked. No documentation could be found from *Google AdMob* stating that they collect location information like *Accuracy*, *Latitude* and *Longitude*, information regarding the system *memory*, if it was a *simulator* instead of a real device, and information about the state of the *battery*. *New Relic*, from what could be found, does not state that it collects information about the *disk*, *memory* and application *orientation*. *Pushwoosh* enumerated the elements they collected, however, there was no evidence from them stating that the *jailbroken* status and *connection* information is collected. Finally, based on what was found, *Startapp* misses to detail that they collect information regarding the *battery*, *root* status, *screen* properties, *memory*, and dozens of elements related to the *network* of the device, in their policy.

To the best effort, it could not be found a disclosure by *Amplitude* and *Segment (Twilio)* of the data they collect from their Android SDKs. No privacy policy or data collection information was found for *AltBeacon* and *Open Telemetry* (however, no tracking behavior from them was found).

Finally, some trackers, like *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...*”. Including these kinds of trackers in government-sponsored apps that restrain the spread of a global pandemic can be, at the very least, controversial.

6.5 Observations Regarding Possible Roots of the Problem

One interesting question is, why do trackers find their way into these critical apps, most of them sponsored by governments, highly influential organizations, universities, or health providers? How is this possible?

Without a doubt, to answer these questions, a focused study would be required; however, based on the information studied, we provide some initial explanations.

SDKs and privacy implications. Developers and privacy teams may not be aware of the privacy implications of including SDKs in their apps. Adding SDKs, especially trackers, involves information flowing from the device to the SDK server (thus, a third party). Therefore, they need to validate that this data exfiltration is adequate and accepted by the user.

Technical complexity. For privacy teams, understanding what trackers do and how they can find this information (for example, following the steps taken in this study) can be too complex or technical. On the other hand, understanding what information is

Tracker	Declares	Coincides	References
AdColony	Policy	No	[3] [2]
Airship	Enumeration	Yes	[8]
AltBeacon	-	-	No Data
Amplitude	Policy	No Data	[13]
AppNext	Policy	No	[17] [16]
Branch	Mix	Yes	[20]
Braze	Enumeration	No	[22] [23]
Bugsnap	Enumeration	Yes	[24]
Flurry	Enumeration	Yes	[49]
Google AdMob	Enumeration	No	[4]
Google Firebase	Enumeration	Yes	[47] [48]
Google Tag Manager	Enumeration	Yes	[85]
Mapbox	Policy	Yes	[64]
Matomo	Enumeration	Yes	[65]
MixPanel	Enumeration	Yes	[67]
New Relic	Enumeration	No	[71]
OneSignal	Enumeration	Yes	[72]
Open Telemetry	-	-	No Data
Pushwoosh	Enumeration	No	[76]
Segment	Policy	No Data	[80]
Splunk MINT	Enumeration	Yes	[81]
Startapp	Policy	No	[82] [83]

TABLE 6.1: Trackers declaration of data collected with their Android SDK.

sensitive, personal, or private under applicable regulations may be outside the scope of development teams.

Tools availability. To the best of our efforts, we could not find open-source tools to help privacy or development teams review the activity performed by the tracker’s code within their apps. This lack of tools may hinder privacy analysis since the unique available approach would be to read the SDK code (if publicly available in, for example, repositories as GitHub or Bitbucket) or, even more complex, resort to reverse engineering mechanisms. As previously stated in Chapter 5, this was one of the main motivations to develop SAPITO.

Communication issues. As in every organization with multiple disparate teams, good communication channels are essential. Here, for example, development teams should make privacy teams aware of the inclusion of SDKs into the apps and what it implies since the privacy impact of that could even be an obstacle to its release or, at least, may demand a revision of the application’s privacy policy. The practice of using *Privacy Champions* in development teams shows promising results [86].

Project priorities and resources. Currently, privacy should be a priority in any software project. Sadly, this does not happen. For example, this lack of attention to privacy was found in startups [27]. However, governments and other international organizations are not startups, and they must take privacy as one of the most important requirements in the systems and applications they develop or sponsor. On the other hand, in jurisdictions where privacy is a priority, resources for it may not be abundant, especially in

the governmental sector.

Privacy statements by SDK providers. Companies behind SDKs (including trackers) want these to be used. Hence, it is in their best interest not to highlight their SDKs' tracking behavior. The common practice is to put the minimum information on their policies simply to comply with applicable law. This information is mostly written in their terms of services, using legal jargon. What ends up happening is that the decision whether to use the tracker or not is taken by developers, which may not have the proper role and legal knowledge to take them [68].

Privacy services. Finally, another cause could be the lack of expert teams providing third-party privacy services with enough understanding of the legal and technical components in every jurisdiction releasing these critical and widely used mobile apps. However, important regulations like *GDPR* and other data protection laws may have helped with data privacy awareness and the creation of new privacy services offered and demanded worldwide.

Chapter 7

Conclusion and Further Work

As described in Chapter 4, **trackers were extensively used in COVID-19 mobile apps**. A total of 58 different trackers were found in the 595 studied apps, some of them containing over 5 trackers. However, it is worth noting that almost one-third of the apps in the collected data set did not include any tracker. In addition, historical research of *Exodus* reports demonstrated that the number of trackers included in the apps did not necessarily decrease throughout the pandemic. Of 102 apps that contained more than one report and a tracker in the first one of these, only 25 apps showed a decrease in the number of contained trackers (and just 6 of these removed all trackers present in the app).

Furthermore, the analysis of the trackers' code allowed us precisely to identify the information being collected and sent to their servers. It was found that in COVID-19 related apps, information shared with third-parties included the *Android Advertisement ID of the device; its specific location and locale configuration; its operating system, model, brand, manufacturer and rooted condition; the status of memory, disk, screen, audio, and battery; network data; and the application where the trackers were included, plus other running apps and processes*. These elements were contrasted with what trackers detail in their data collection disclosure documentation and found that, in some cases, there was not a complete disclosure of all items being collected. Moreover, the use of **push notification** services also has the potential to disclose sensitive information related to the health status of app users to the providers of these services, depending on the content of the messages.

Chapter 6 discusses the possible consequences of including trackers and push notification services for COVID-19 apps. It was concluded that, under certain conditions, sensitive information related to users' health could be disclosed to third parties, **potentially violating applicable regulations**. For example, since trackers collect the *app ID*, in case this app were used to enforce quarantine or to track the disease evolution in a user, they could **infer that the app user was infected with COVID-19**. Moreover, since they collect *location* information, they could identify **where infected users were and the places they visited** (including identifying in this manner their health provider). Another example of disclosure to third parties could be when the users' positive **infection status** is notified using push notification services. Additionally, some trackers openly disclose that **data collected could be transferred or sold** to other parties. Therefore, the

impact on the users increases when these trackers are used. Finally, since trackers were included in apps with millions of downloads, the number of affected users is extremely large (hundreds of millions).

Data protection initiatives were also discussed, coming mainly from the government sector, oriented toward providing guidelines and requirements concerning the privacy of citizens' data manipulated by the applications studied. Despite a vast amount of research focused on the ecosystem of Android trackers, industry and government sectors seem to be unaware of the impact these libraries may have on data protection when included in their apps. It is clear that privacy policies and data protection impact assessments have room for improvement.

This kind of research of trackers in the scope of COVID-19 apps can be extrapolated to application ecosystems like financial and payments, where *bank secrecy* could be violated if practical information regarding the financial activities of the app users is transferred to third parties. It would also be interesting to keep exploring the health domain, where apps could allow trackers to collect information about users without their consent.

As commented in Chapter 6, further study would be needed to identify why trackers are included in critical apps. Are their services strictly necessary for the apps? Are there safer alternatives? Even if no health data is directly processed, the health status of the apps users could be inferred, with some data aggregation.

Similarly, push notification services could violate data protection regulations, since sensitive information is transmitted in plaintext to third parties. Developers should publicly inform the users of this data transfer method (not only in the privacy policy) and, ideally, obtain their consent to use it.

Research focused on why trackers end in critical apps has the potential to identify mitigating measures encouraging best practices regarding privacy analyzes. As an example, the execution of data protection impact assessments in every future application developed or sponsored by governments must be encouraged, including the analysis of SDK usage in the apps. Based on what was seen, just a few jurisdictions studied the impact of their inclusion in COVID-19 apps.

On the other hand, trackers that explicitly indicate that they segment users and sell the collected information to other parties must be avoided at all costs in these apps.

Also, a prototype of **an open-source tool to analyze SDKs in Android apps was developed (SAPITO)**, especially helpful to detect tracking behavior from these packages.

In its current status, the tool SAPITO has some limitations. First, it can detect trackers present in the application, but it does not check whether they are activated or called at any point. As long as they are dormant and do not have an API key associated, they are not dangerous. Besides these false positives, there may also be some false negatives, since the dangerous behavior detection rules are based on what has been observed in the trackers analyzed. Thus, there may be cases that were not found in this study and are not included in those detection rules.

Additionally, to study tracking behavior, the analysis done from finding the instruction where data is collected to the moment it is sent to the third-party server is quite complex (the binary usually has thousands of classes and dozen of thousand of methods, most of them obfuscated). Hence, it would then be helpful to add to the developed tool a module to manually, or even better, automatically, build a diagram following the path of the harvested data, such as a data flow diagram, a class diagram, or a call graph.

A different line of work is to extend SAPITO with the ability to process iOS applications and to provide support for the analysis of trackers in the web landscape since most trackers analyzed in this research provide web SDKs as well [21, 14, 70, 12, 73].

Bibliography

- [1] Anglemyer A et al. “Digital contact tracing technologies in epidemics: a rapid review”. In: *Cochrane Database Syst Rev* (2020).
- [2] *AdColony Consumer Policy*. <https://www.adcolony.com/consumer-privacy/>. Accessed: 2022-09-29.
- [3] *AdColony Privacy Policy*. <https://www.adcolony.com/privacy-policy/>. Accessed: 2022-09-29.
- [4] *AdMob Data Collection*. <https://support.google.com/admob/answer/9755590>. Accessed: 2022-09-29.
- [5] AWO Agency. “GOVERNMENT RESPONSES TO THE COVID-19 PANDEMIC”. In: (2021).
- [6] AWO Agency. “Report on the privacy risks of COVID-19 software”. In: (2020).
- [7] Nadeem Ahmed et al. “A Survey of COVID-19 Contact Tracing Apps”. In: *IEEE Access* 8 (2020), pp. 134577–134601. DOI: [10.1109/ACCESS.2020.3010226](https://doi.org/10.1109/ACCESS.2020.3010226).
- [8] *Airship Data Collection*. <https://docs.airship.com/platform/android/data-collection/>. Accessed: 2022-09-29.
- [9] Zainab Alfayez, Nabaa Al-Sinayyid, and Sadeq AL-Ameri. “Mobile Applications Developed By Arab Countries In Response To Covid-19: A review”. In: *Journal of Information System and Technology Management* 6 (Sept. 2021), pp. 200–211. DOI: [10.35631/JISTM.622016](https://doi.org/10.35631/JISTM.622016).
- [10] Amal Awadalla Ali et al. “Contact Tracing Apps for COVID-19: Access Permission and User Adoption”. In: *2020 7th International Conference on Behavioural and Social Computing (BESC)*. 2020, pp. 1–7. DOI: [10.1109/BESC51023.2020.9348327](https://doi.org/10.1109/BESC51023.2020.9348327).
- [11] Kevin Allix et al. “AndroZoo: Collecting Millions of Android Apps for the Research Community”. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR ’16. Austin, Texas: ACM, 2016, pp. 468–471. ISBN: 978-1-4503-4186-8. DOI: [10.1145/2901739.2903508](https://doi.org/10.1145/2901739.2903508). URL: <http://doi.acm.org/10.1145/2901739.2903508>.
- [12] *Amplitude Analytics Browser*. www.npmjs.com/package/@amplitude/analytics-browser. Accessed: 2022-09-19.
- [13] *Amplitude Privacy*. <https://amplitude.com/privacy>. Accessed: 2022-09-29.

- [14] *Analytics.js 2.0 Source*. <https://segment.com/docs/connections/sources/catalog/libraries/website/javascript/#analytics-js-2-0-source>. Accessed: 2022-09-19.
- [15] *Androguard*. <https://github.com/androguard/androguard>. Accessed: 2022-09-19.
- [16] *Appnext DPA*. <https://www.appnext.com/dpa/>. Accessed: 2022-09-29.
- [17] *Appnext Policy*. <https://www.appnext.com/policy.html>. Accessed: 2022-09-29.
- [18] Reuben Binns et al. "Third Party Tracking in the Mobile Ecosystem". In: *Proceedings of the 10th ACM Conference on Web Science*. WebSci '18. Amsterdam, Netherlands: Association for Computing Machinery, 2018, pp. 23–31. ISBN: 9781450355636. DOI: 10.1145/3201064.3201089. URL: <https://doi.org/10.1145/3201064.3201089>.
- [19] Kirsten Bock et al. "Data Protection Impact Assessment for the Corona App". In: *SSRN Electronic Journal* (Jan. 2020). DOI: 10.2139/ssrn.3588172.
- [20] *Branch Privacy Policy*. <https://branch.io/policies/privacy-policy/#privacy-what-information-does-branch-collect-from-our-services>. Accessed: 2022-09-29.
- [21] *Branch Web Full Reference*. <https://help.branch.io/developers-hub/docs/web-full-reference>. Accessed: 2022-09-19.
- [22] *Braze Data Collection*. https://www.braze.com/docs/developer_guide/platform_integration_guides/android/google_play_privacy/. Accessed: 2022-09-29.
- [23] *Braze SDK Data Collection*. https://www.braze.com/docs/user_guide/data_and_analytics/user_data_collection/sdk_data_collection. Accessed: 2022-09-29.
- [24] *Bugsnag Data Collection*. <https://docs.bugsnag.com/platforms/android/automatically-captured-data/>. Accessed: 2022-09-29.
- [25] *Capillary*. <https://github.com/google/capillary>. Accessed: 2022-10-02.
- [26] Davide Caputo et al. "You Can't Always Get What You Want: Towards User-Controlled Privacy on Android". In: *IEEE Transactions on Dependable and Secure Computing* (Jan. 2022), pp. 1–1. DOI: 10.1109/TDSC.2022.3146020.
- [27] Wenhong Chen et al. "'As We Grow, It Will Become a Priority': American Mobile Start-Ups' Privacy Practices". In: *American Behavioral Scientist* 62.10 (2018), pp. 1338–1355. DOI: 10.1177/0002764218787867. eprint: <https://doi.org/10.1177/0002764218787867>. URL: <https://doi.org/10.1177/0002764218787867>.
- [28] Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. "Contact Tracing Mobile Apps for COVID-19: Privacy Considerations and Related Trade-offs". In: *ArXiv abs/2003.11511* (2020).

- [29] Andrea Continella et al. "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, Feb. 2017.
- [30] *Coronavirus disease COVID-19*. www.who.int/health-topics/coronavirus. Accessed: 2022-09-15.
- [31] *Covid Safe Privacy Policy*. <https://cert-app.be/en/privacy-covidsafe.html>. Accessed: 2022-09-30.
- [32] *Covid Tracing Tracker*. https://docs.google.com/spreadsheets/d/1ATaIAS08KtZMx_zJREoOvFh0nmB-sAqJ1-CjVRSC0w. Accessed: 2022-10-05.
- [33] *COVID-19 Contact tracing: data protection expectations on app development*. <https://ico.org.uk/media/for-organisations/documents/2617676/ico-contact-tracing-recommendations.pdf>. Accessed: 2022-10-04.
- [34] *COVID-19 Dashboard, Johns Hopkins University*. www.arcgis.com/apps/dashboards/bda7594740fd40299423467b48e9ecf6. Accessed: 2022-09-15.
- [35] *COVID19 - DXB Smart App Privacy Policy*. <https://www.dha.gov.ae/en/privacy-policy>. Accessed: 2022-09-30.
- [36] Paul-Olivier Dehaye and Joel Reardon. "Proximity Tracing in an Ecosystem of Surveillance Capitalism". In: *Proceedings of the 19th Workshop on Privacy in the Electronic Society*. WPES'20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 191–203. ISBN: 9781450380867. DOI: 10.1145/3411497.3420219. URL: <https://doi.org/10.1145/3411497.3420219>.
- [37] *Device Fingerprinting*. <https://blog.getsocial.im/device-fingerprinting-for-mobile-attribution/>. Accessed: 2022-10-03.
- [38] *DPIA Covid Tracker App*. <https://github.com/HSEIreland/covidtracker-documentation/blob/master/documentation/privacy/>. Accessed: 2022-09-30.
- [39] *DPIA CovidCert*. <https://covid-19.hscni.net/covidcert-ni-mobile-app/>. Accessed: 2022-09-30.
- [40] *DPIA Koronavirus*. https://www.koronavirus.hr/uploads/Stop_COVID_19_Data_Protection_Impact_Assesment_Summary_2020_11_16_58dea76816.pdf. Accessed: 2022-09-30.
- [41] *DPIA NHS*. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1028998/NHS_COVID_19_App_DPIA.pdf. Accessed: 2022-09-30.
- [42] *DPIA NZ COVID Tracer*. https://www.health.govt.nz/system/files/documents/pages/contact_tracing_app_pia_for_release_10_final.pdf. Accessed: 2022-09-30.
- [43] *DPIA StopCOVID NI*. <https://covid-19.hscni.net/wp-content/uploads/2020/10/DPIA-for-StopCOVID-NI-Proximity-App-14.10.pdf>. Accessed: 2022-09-30.

- [44] *Ehteraz Privacy Policy*. <https://portal.moi.gov.qa/met2/privacyehteraz.html>. Accessed: 2022-09-30.
- [45] *Exodus Standalone*. <https://github.com/Exodus-Privacy/exodus-standalone>. Accessed: 2022-09-20.
- [46] *Exodus static analysis*. https://exodus-privacy.eu.org/en/post/exodus_static_analysis/. Accessed: 2022-09-19.
- [47] *Firebase Data Collection 1*. <https://support.google.com/firebase/answer/9234069>. Accessed: 2022-09-29.
- [48] *Firebase Data Collection 2*. <https://support.google.com/firebase/answer/9268042>. Accessed: 2022-09-29.
- [49] *Flurry Data Disclosure*. <https://www.flurry.com/blog/google-play-data-disclosure-requirements/>. Accessed: 2022-09-29.
- [50] *Google Play Safety Section*. <https://android-developers.googleblog.com/2021/07/new-google-play-safety-section.html>. Accessed: 2022-09-29.
- [51] *gplaydl*. <https://github.com/rehmatworks/gplaydl>. Accessed: 2022-09-20.
- [52] Majid Hatamian et al. "A Privacy and Security Analysis of Early-Deployed COVID-19 Contact Tracing Android Apps". In: *Empirical Softw. Engg.* 26.3 (May 2021). ISSN: 1382-3256. DOI: [10.1007/s10664-020-09934-4](https://doi.org/10.1007/s10664-020-09934-4). URL: <https://doi.org/10.1007/s10664-020-09934-4>.
- [53] *Hayat Eve Sigar Privacy Policy*. https://hayatevesigar.saglik.gov.tr/gizlilik_politikasi_eng_index_V2.html. Accessed: 2022-09-30.
- [54] Yongzhong He et al. "Dynamic privacy leakage analysis of Android third-party libraries". In: *Journal of Information Security and Applications* 46 (2019), pp. 259–270. ISSN: 2214-2126. DOI: <https://doi.org/10.1016/j.jisa.2019.03.014>. URL: <https://www.sciencedirect.com/science/article/pii/S2214212618304356>.
- [55] Nakamoto I et al. "A QR Code-Based Contact Tracing Framework for Sustainable Containment of COVID-19: Evaluation of an Approach to Assist the Return to Normal Activity". In: *JMIR Mhealth Uhealth* (2020).
- [56] *Instant Apps*. <https://developer.android.com/topic/google-play-instant>. Accessed: 2022-09-23.
- [57] Samhi J, Allix K, and Bissyandé T Fand Klein J. "A first look at Android applications in Google Play related to COVID-19". In: *Empirical Software Engineering* 26 (4 2021).
- [58] *jadx - Dex to Java decompiler*. <https://github.com/skylot/jadx>. Accessed: 2022-09-19.
- [59] Kollnig Konrad et al. "A Fait Accompli? An Empirical Study into the Absence of Consent to Third-Party Tracking in Android Apps". In: (June 2021).

- [60] *Letter to FTC*. <https://consumerwatchdog.org/resources/LtrFTCFinal.pdf>. Accessed: 2022-09-12.
- [61] Xing Liu et al. "Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem". In: *IEEE Transactions on Mobile Computing* 19.5 (2020), pp. 1184–1199. DOI: [10.1109/TMC.2019.2903186](https://doi.org/10.1109/TMC.2019.2903186).
- [62] Dieter M et al. "Pandemic platform governance: Mapping the global ecosystem of COVID-19 response apps". In: *Internet Policy Review* 10 (3 2021).
- [63] Azad MA et al. "A First Look at Privacy Analysis of COVID-19 Contact-Tracing Mobile Applications". In: *IEEE Internet Things J* (2020).
- [64] *Mapbox Privacy*. <https://www.mapbox.com/legal/privacy>. Accessed: 2022-09-29.
- [65] *Matomo Data Collection*. https://matomo.org/faq/general/faq_18254/. Accessed: 2022-09-29.
- [66] *MIT Technology Review Covid Tracing Tracker*. <https://www.technologyreview.com/2020/12/23/1015557/covid-apps-contact-tracing-suspended-replaced-or-relaunched/>. Accessed: 2022-10-03.
- [67] *Mixpanel Data Collection*. <https://help.mixpanel.com/hc/en-us/articles/115004613766-Default-Properties-Collected-by-Mixpanel#android/>. Accessed: 2022-09-29.
- [68] Tahaei Mohammad and Vaniea Kami. "'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. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380959. DOI: [10.1145/3411763.3451805](https://doi.org/10.1145/3411763.3451805). URL: <https://doi.org/10.1145/3411763.3451805>.
- [69] Sergio Nesmachnow and Santiago Iturriaga. "Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay". In: *Supercomputing*. Ed. by Moisés Torres and Jaime Klapp. Cham: Springer International Publishing, 2019, pp. 188–202. ISBN: 978-3-030-38043-4.
- [70] *New Relic Browser Monitoring*. <https://newrelic.com/platform/browser-monitoring>. Accessed: 2022-09-19.
- [71] *New Relic Privacy*. <https://docs.newrelic.com/docs/mobile-monitoring/newrelic-mobile/get-started/security-mobile-apps>. Accessed: 2022-09-29.
- [72] *OneSignal Data Collection*. <https://documentation.onesignal.com/docs/data-collected-by-the-onesignal-sdk>. Accessed: 2022-09-29.
- [73] *OneSignal Web Push*. <https://onesignal.com/webpush>. Accessed: 2022-09-19.
- [74] *Pedulilindungi Privacy Policy*. <https://www.pedulilindungi.id/kebijakan-privasi-data?lang=en>. Accessed: 2022-09-30.
- [75] *Privacy – a fundamental right*. https://edps.europa.eu/data-protection/data-protection_en. Accessed: 2022-09-30.

- [76] *Pushwoosh Data Collection*. <https://help.pushwoosh.com/hc/en-us/articles/360015945252-What-user-data-does-Pushwoosh-collect->. Accessed: 2022-09-29.
- [77] *Pushwoosh Push Notification Privacy*. <https://blog.pushwoosh.com/blog/how-to-work-with-sensitive-data-if-you-want-to-use-push-notifications-2/>. Accessed: 2022-09-24.
- [78] Abbas Razaghpanah et al. "Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem". In: *NDSS*. 2018.
- [79] *SAPITO*. <https://gitlab.fing.edu.uy/gsi/sapito>. Accessed: 2023-11-12.
- [80] *Segment (Twilio) Privacy*. <https://www.twilio.com/legal/privacy>. Accessed: 2022-09-29.
- [81] *Splunk MINT Data Collection*. <https://docs.splunk.com/Documentation/Mint/1.0/ProductOverview/AboutSplunkMINTdatacollection>. Accessed: 2022-09-29.
- [82] *Startapp Privacy*. <https://www.start.io/privacy-faq/>. Accessed: 2022-09-29.
- [83] *Startapp Privacy Policy*. <https://www.start.io/policy/privacy-policy-site/>. Accessed: 2022-09-29.
- [84] Ryan Stevens et al. "Investigating User Privacy in Android Ad Libraries". In: 2012.
- [85] *Tag Manager Data Collection*. <https://support.google.com/tagmanager/answer/9323295>. Accessed: 2022-09-29.
- [86] Mohammad Tahaei, Alisa Frik, and Kami Vaniea. "Privacy Champions in Software Teams: Understanding Their Motivations, Strategies, and Challenges". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380966. DOI: 10.1145/3411764.3445768. URL: <https://doi.org/10.1145/3411764.3445768>.
- [87] Gioacchino Tangari et al. "Mobile health and privacy: Cross sectional study". In: *BMJ* 373 (June 2021), n1248. DOI: 10.1136/bmj.n1248.
- [88] Kouliaridis V et al. "Dissecting contact tracing apps in the Android platform". In: *PLoS One* (2021).
- [89] Jice Wang et al. "Understanding Malicious Cross-library Data Harvesting on Android". In: *USENIX Security Symposium*. 2021.
- [90] Liu Wang et al. "Beyond the virus: a first look at coronavirus-themed Android malware". In: *Empirical Software Engineering* 26.4 (2021). ISSN: 1573-7616. DOI: 10.1007/s10664-021-09974-4. URL: <https://doi.org/10.1007/s10664-021-09974-4>.
- [91] Liu Wang et al. *Beyond the Virus: A First Look at Coronavirus-themed Mobile Malware*. 2021. arXiv: 2005.14619 [cs.CR].

- [92] Haohuang Wen et al. "A Study of the Privacy of COVID-19 Contact Tracing Apps". In: *Security and Privacy in Communication Networks*. Ed. by Noseong Park et al. Cham: Springer International Publishing, 2020, pp. 297–317. ISBN: 978-3-030-63086-7.
- [93] Wenjia Wu et al. "Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers". In: *IEEE Access* 4 (2016), pp. 8073–8083. DOI: [10.1109/ACCESS.2016.2626395](https://doi.org/10.1109/ACCESS.2016.2626395).
- [94] F Yang, L Heemsbergen, and R Fordyce. "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?" In: *Media International Australia* (2021).
- [95] Zenodo, CERN. www.eui.eu/Research/Library/ResearchGuides/Economics/Statistics/DataPortal/Zenodo. Accessed: 2022-09-20.
- [96] Sophia L. Zhou et al. "Lessons on mobile apps for COVID-19 from China". In: *Journal of Safety Science and Resilience* 2.2 (2021), pp. 40–49. ISSN: 2666-4496. DOI: <https://doi.org/10.1016/j.jnlssr.2021.04.002>. URL: <https://www.sciencedirect.com/science/article/pii/S2666449621000128>.

Appendix A

List of COVID-19 Android applications analyzed

In this Appendix, we list the applications that were studied in our research, pointing to the country, region or jurisdiction where it was intended.

Country	App ID	Country	App ID
Albania	dgca.verifier.app.covidpassandroidal	Algeria	com.covid19_algeria
Andorra	ad.saas.andorrasalutapp	Angola	com.bullray.covid
Argentina	com.c19.sl	Argentina	ar.gob.coronavirus
Armenia	am.gov.covid19	Armenia	com.sylex.armed
Aruba	org.dvgapp.tourist	Australia	org.beatcovid19now.application
Australia	au.gov.health.covid19	Australia	au.gov.health.covidsafe
Australia	au.gov.act.health.checkin	Australia	au.gov.nt.health.checkin
Australia	au.gov.nsw.service	Australia	au.gov.qld.checkin
Australia	au.gov.sa.my	Australia	au.gov.tas.checkin
Australia	au.gov.vic.service.digitalwallet.citizen	Australia	au.gov.wa.digital.service.mobile.servicewa...
Austria	at.roteskreuz.stopcorona	Austria	at.gv.brz.wallet
Azerbaijan	az.gov.etabib	Azerbaijan	com.uptodate
Azerbaijan	az.gov.my	Bahrain	bh.bahrain.corona.tracker
Bangladesh	bd.com.cmed.agent	Bangladesh	com.bs.ccc
Bangladesh	com.shohoz.tracer	Bangladesh	com.codersbucket.surokkha_app
Barbados	com.xafe.bb.bimSAFE	Barbados	io.xafe.bb.bimshield
Belgium	be.sciensano.coronalert	Belgium	be.fgov.ehealth.DGC
Belize	com.idealabstudios.mohwpassapp	Bermuda	org.wehealth.exposure
Bhutan	bt.gov.moh.druktrace	Bolivia	com.sedes.bsana
Bolivia	com.agnetic.coronavirusapp	Botswana	com.gov.bw.iam
Brazil	br.com.novetech.monitoracoronavirus	Brazil	com.brazil.corona_brazil_check
Brazil	com.apptodeolho	Brazil	br.gov.datasus.guardioes
Brazil	br.gov.datasus.cnsdigital	British Virgin Is.	px.mw.android.aihealth.patient.bhslive...
Brunei	egnc.moh.bruhealth	Bulgaria	bg.government.virusafe
Bulgaria	dcc.check.bg	Burkina Faso	com.sante.dioula
Burkina Faso	com.sante.covid2	Burkina Faso	bf.diagnoseme.fasocivic
Cambodia	com.khmer_vacc	Canada	ca.gc.cbsa.coronavirus
Canada	ca.gc.hcsc.canada.covid19	Canada	ca.bc.gov.health.hlbc.COVID19
Canada	ca.albertahealthservices.contacttracing	Canada	ca.gc.hcsc.canada.stopcovid
Canada	ca.quebec.vaxilecteurandroid	Canada	com.govmb.immunizationcard
Canada	ca.bc.gov.vaxcheck	Canada	ca.ohri.immunizeapp
Canada	ca.ontario.verify	Canada	ca.quebec.vaccandroid
Canada	com.govmb.immunizationrecord	Cape Verde	com.devtrust.cv.comvida
Cape Verde	cv.nosi.app.nhacard.validator	Caribbean	org.carpha.caritrivhealth
Cayman Islands	com.cerner.iris.play	Chile	cl.gob.digital.coronapp
China	com.systoon.dongaotoon	China	mo.gov.ssm.Macao_Health_Codev2
Colombia	appinventor.ai_edw.pores.SALUDPROY...	Colombia	co.gov.cali.calivallecorona
Colombia	co.gov.ins.guardianes	Colombia	com.mocaplatform.cuidemonos
Colombia	icesi.uccare	Congo, Dem. Rep.	id.jofarsystems.kodine.dev

TABLE A.1: List of studied COVID-19 Android applications (Part 1/5).

Country	App ID	Country	App ID
Congo, Dem. Rep.	com.rdctrack.covid1	Cook Islands	ck.gov.cooksafeplus
Costa Rica	com.ccss.expedienteunico	Costa Rica	gov.cr.enx.v3
Croatia	hr.miz.evidencijakontakata	Croatia	hr.akd.dzp
Cuba	club.postdata.covid19cuba	Cuba	cu.sld.COVID_19_InfoCU
Curacao	com.dushistay.app	Cyprus	cy.gov.dmrld.covtracer
Cyprus	cy.gov.eudcc.app.verifier_lite.android	Cyprus	cy.gov.eudcc.app.wallet.android
Czech Republic	covid.czstatistika.covidcz	Czech Republic	com.appsisto.coronaviruscovid19
Czech Republic	cz.covid19cz.erouska	Czech Republic	cz.seznam.mapy
Czech Republic	cz.nakit.eocko.validate	Czech Republic	cz.nakit.eocko.wallet
Denmark	com.netcompany.smittestop_exposure_noti...	Denmark	com.synlab.covidpass
Denmark	dk.sum.ssicpas	Dominican Rep.	com.optic.covdr
Ecuador	ec.gob.gobiernoelectronico.coronavirus	Ecuador	com.phuyusalud.movil
Ecuador	ec.gob.asi.android	Egypt	com.egypt.hajj
Egypt	com.vio.ettammen	Egypt	eg.com.eserve.sehatmisr
Egypt	get.pid.egypt.health.passport	Eritrea	org.undp.er.health
Estonia	ee.tehik.hoia	Ethiopia	com.twinsoftplc.covidtigray
Ethiopia	et.gov.moh.oppia.covid	Ethiopia	com.ewenet.debo
Europe	eu.europa.publications.reopeneu	Europe	com.secutex.blockchain.healthngo
Europe	pt.incm.eudcc.app.lite	Europe	com.nocovid19.su
Fiji	fj.gov.carefiji	Finland	fi.thl.koronahaavi
Finland	fi.thl.koronatodistus	France	com.ambulis.aphm.covid
France	fr.aphp.covidom	France	fr.gouv.android.stopcovid
France	com.ingroupe.verify.anticovid	France	com.vogo.easycov
Georgia	gov.georgia.novid20	Georgia	com.moh.geehealth
Germany	de.labuniq.medicover	Germany	de.rki.coronadatenspende
Germany	com.coronacheck.haugxhaug.testyourcorona	Germany	de.bssd.covid19
Germany	com.reactnativeapp	Germany	de.kreativzirkel.coronika
Germany	de.rki.coronawarnapp	Germany	de.rki.covpass.app
Germany	de.zollsoft.impfapp	Ghana	com.moc.gh
Gibraltar	com.gha.covid.tracker	Gibraltar	com.gha.gibraltarcovidapp
Greece	com.docandu.checker	Greece	org.pathcheck.gr.bt
Greece	gr.gov.dcc.mini	Greece	gr.gov.dcc.wallet
Grenada	com.outsys.covidsupportapp	Guatemala	com.intelligent.alertaguate
Honduras	com.doctor1847.unah.edu.hn	Hong Kong	com.compathnion.equarantine
Hong Kong	hk.gov.ogcio.leavehomesafe	Hong Kong	hk.gov.ogcio.covidresultqrscanner.full
Hungary	com.hkr	Hungary	hu.gov.virusradar
Hungary	hu.gov.eeszt.mgw.covidpassportcontrol	Iceland	is.landlaeknir.rakning
Iceland	is.landlaeknir.skanni	India	com.negd.ayushfeedback
India	com.NIC.covid19	India	in.gov.surveyofindia.sahyog
India	com.innovaccer.testyourselfgoa	India	com.innovaccer.testyourselfpuddu
India	odisha.gov.covid19	India	com.ksbvvirtualq
India	com.bsafetracking	India	com.developmentlogics.patientgeotracker
India	com.pixxonai.covid19	India	com.bmc.qrtwatch
India	com.covid19.dgmup	India	com.homequarantine
India	com.intutrack.covidtrack	India	com.pixxonai.covid19wb
India	com.pratikthorath.coronatraccker	India	www.facetagr.com.cobuddy
India	org.nic.covidcarekannur	India	com.flowace.saiyam
India	ai.zini.covoid	India	com.entrolabs.pharmacyap
India	hr.gov.covid19.sahayak	India	com.app_release.covid19finalcourse
India	com.app.coronabandungkab	India	com.delhi.covidcare
India	com.entrolabs.apcovid19	India	com.gmda.govid19
India	com.ocac.covidodisha	India	com.qkopy.prdkerala
India	com.stucare.ayush	India	com.tenten.coronawarriors
India	com.tsstate.citizen	India	gov.mizoram.mcovid19
India	in.gaia.smartcadre.agra	India	in.gov.chhattisgarh.cova
India	in.mygov.mobile	India	in.gov.wb.wbreliefund
India	co.cellapp.bharatpurcovid	India	com.mahakavach
India	nic.goi.aarogyasetu	India	com.krsac.drawshapefile
India	com.cosafe.android	India	com.gcc.smartcity
India	in.smc.covidout	India	com.allsoft.corona
India	com.digilocker.android	India	in.gov.umang.negd.g2c
Indonesia	com.kemendes.inahac	Indonesia	com.LawanCovid19FC19S
Indonesia	com.deepcovid19	Indonesia	com.sepuluh.rumahaman
Indonesia	id.go.jabarprov.pikobar	Indonesia	id.simpkb.guruberbagi
Indonesia	id.anwarhasan.presensiwfh	Indonesia	com.telkom.tracencare
International	com.solucionesun.pmovil	International	com.daon.glide.person.android
International	cz.nmbrno.covid	International	com.preemeyou.covid19
International	com.webmd.android	International	de.xikolo.openwho
International	org.un.corona	International	org.who.infoapp
International	org.who.WHOA	International	com.who_mobile2020
International	org.who.COVIDKAYA	International	org.who.LENA
International	com.app.corona360	International	com.bloomrealitysodi
International	world.coalition.app	International	com.dhis2
International	com.aokpass.aokpasspilot	International	com.tento.wallet
International	eu.yourpass.wallet	International	org.thecommonsproject.android.commonpass
Iran	co.health.covid	Iraq	com.y.twekl.CoronaVaccine
Iraq	com.twekl.khg	Ireland	com.maithu.transplantbuddy.covid19
Ireland	com.covidtracker.hse	Ireland	ie.healthpassportireland.patient
Ireland	com.gnomon.ehealthpass.ie	Israel	com.hamagen
Israel	org.track.virus	Israel	com.moh.alert.ramzor
Italy	it.lispa.sire.app.mobile.allertalom	Italy	com.intellicare.covid
Italy	com.siciliasicura.app	Italy	com.tommasomauriello.autocertificazione...
Italy	com.salvagente.info	Italy	it.adilife.covid19.app
Italy	it.adl.aslroma3.covid19.app	Italy	it.ministerodellasalute.immuni

TABLE A.2: List of studied COVID-19 Android applications (Part 2/5).

Country	App ID	Country	App ID
Italy	it.softmining.projects.covid19.savelifestyle	Italy	it.regione.sardegna.autorizzazionecovid19
Italy	srl.digit.diary	Italy	it.ministerodellasalute.verificaC19
Italy	it.pagopa.io.app	Jamaica	com.jamcovid19
Japan	jp.co.emergency.followup	Japan	jp.tokyo.chofu.city.w2.covid19
Japan	jp.go.mhlw.covid19radar	Japan	jp.go.digital.vrs.vpa
Jersey	com.governmentofjersey.jerseycovidalert	Jordan	com.menaitech.NCSCM
Jordan	jo.gov.moh.aman	Jordan	com.moddee.sanad
Kazakhstan	kz.citysoft.smartastana	Kazakhstan	asyq.curs.kz
Kazakhstan	kz.nitec.bizbirgemiz	Kazakhstan	kz.mobile.mgov
Kenya	org.medicmobile.webapp.mobile.surveill...	Kenya	com.mhealthkenya.dm.mohkenya
Korea, South	com.mohw.corona	Korea, South	kr.go.safekorea.sqsmo
Korea, South	kr.go.safekorea.sqsm	Korea, South	com.caragon.caragon.coronamap
Korea, South	com.bienciel.centralarm	Korea, South	com.tina3d.corona100m
Korea, South	kr.go.kdca.coov	Kuwait	com.healthcarekw.app
Kuwait	com.mohkuwait.immune	Kyrgyzstan	kg.cdt.stopcovid19
Laos	com.gov.mpt.laokyc	Laos	info.laoscovid19.laoscovid19
Latin America	org.bvsalud.ebluinfo	Latvia	lv.spkc.gov.apuricovid
Latvia	lv.verification.dgc	Lebanon	com.apps2you.MOPH
Lebanon	com.moph.dgcverifier	Liberia	com.tuma.libtravel
Libya	ly.com.tmc.covid19_libya	Libya	com.speetar.app
Lithuania	com.lympo.covid19	Lithuania	lt.nvsc.coronawarnapp
Lithuania	dga.verifier.lt.app.android	Luxembourg	lu.etat.ci.dcc.android
Malaysia	my.gov.onegovappstore.mysejahtera	Malaysia	com.gov.mcmc.projectctatur
Malaysia	my.gov.onegovappstore.mytrace	Malaysia	com.mbks.qmunity
Malaysia	com.sains.safetrace	Malaysia	com.app.jekajjohor
Malaysia	com.qualitas.covidmy	Malaysia	my.gov.onegovappstore.healthcertverifier
Malaysia	sel.main.selangkah	Maldives	mv.gov.mohiru
Mali	io.agetcmali.soscovid	Mali	org.medicmobile.webapp.mobile.covid...
Malta	mt.gov.dp3t	Malta	mt.gov.CovPassMalta
Mauritius	net.epione.patient	Mauritius	mu.mt.healthapp
Mexico	mx.gob.www	Mexico	gov.mx.yuc.enx.v3
Mexico	covid19.cuernavaca	Mexico	mx.gob.tamaulipas.covid19
Mexico	mx.covidradar.radar	Mexico	com.covid19.cgig
Mexico	gov.mx.coa.enx.v3	Mexico	gov.mx.jal.enx.v3
Mexico	gov.mx.pue.enx.v3	Mongolia	gov.mn.enx
Mongolia	mn.callpro.shuurkhai	Morocco	covid.trace.morocco
Myanmar	com.co.vi.d	Myanmar	mm.org.mcf.app001
Namibia	com.globalgenecorp.namcotrace	Nepal	com.nhr.healthtrackernepal
Nepal	com.iclick.covidnew	Nepal	np.com.naxa.covid19
Nepal	org.prixa.p5covidtracker	Nepal	co.cellapp.bardghatcovid19
Nepal	co.cellapp.lgcovid19	Nepal	co.cellapp.lumbinicovid
Netherlands	com.umcutrecht.covapp	Netherlands	nl.rijksoverheid.ctr.holder
Netherlands	com.umcutrecht.bcg	Netherlands	nl.lumc.covidradar
Netherlands	nl.focuscura.beeldbelapp	Netherlands	com.everywhereim.cocorico
Netherlands	nl.rijksoverheid.en	New Zealand	nz.govt.health.covidpassverifier
New Zealand	health.webtools.awhina.prod	New Zealand	nz.govt.health.covidtracer
Nigeria	com.appcraftng.android5e7dbb5f7e3c4	Nigeria	com.usa.lancorhealthcheck
North Macedonia	rocks.gorjan.covid19vlada	North Macedonia	mk.gov.koronavirus.stop
North Macedonia	com.mkwallet.mk	Northern Ireland	net.hscni.covidcertni
Northern Ireland	net.hscni.covidtracker	Norway	no.fhi.KoronasertifikatKontrollapp
Norway	no.fhi.smittestopp_exposure_notification	Norway	no.simula.smittestopp
Oman	com.emushrif.hmushrif	Oman	om.gov.moh.tarassudapplication
Pakistan	com.passtrack.nitb.gov.pk	Pakistan	pk.gov.nadra.nims.certificate
Pakistan	com.sapphire.HealthMonitoringPDMA	Pakistan	sehat.check
Pakistan	com.sapphire.HealthAssessmentPDMA	Pakistan	com.edu.aku.akuhccheck
Pakistan	com.nap_pakistan.app	Pakistan	covid19care.virus.coronavirus.corona.sick...
Pakistan	pk.pitb.gov.covidtrackerlahore	Pakistan	pk.pitb.gov.rahbar
Pakistan	corona.tracking.system	Pakistan	com.govpk.covid19
Pakistan	in.gov.punjab.cova	Panama	pa.gob.protegete
Paraguay	py.gov.mitic.appcovid19	Peru	pe.gob.regionsanmartin.coronaish
Peru	peruentusmanos.gob.pe	Peru	com.salva.us
Peru	corona.arequipa	Philippines	io.ionic.vaxcertph.verifier
Philippines	com.end.cov	Philippines	ph.staysafe.mobileapp
Philippines	com.voxptech.rc143	Philippines	com.dxfom.ph
Poland	com.symptomate.mobile	Poland	pl.nask.mobywatel
Poland	pl.nask.droid.kwarantannadomowa	Poland	pl.gov.mc.protegosafe
Poland	pl.gov.cez.sws	Poland	pl.gov.cez.mojekip
Portugal	com.vost.covid19mobile	Portugal	ft.inesctec.stayaway
Qatar	com.droobihealth.corona	Qatar	com.moi.covid19
Romania	ro.sts.dcc	Russia	com.minsvyaz.gosuslugi.exposurenotific...
Russia	com.programmisty.emiasapp	Russia	com.askgps.personaltrackerround
Russia	ru.mos.socmon	Russia	com.dawsoftware.contacttracker
Russia	com.minsvyaz.gosuslugi.stopcorona	Rwanda	rw.gov.rbc.rbcc19
Saint Lucia	org.stlucia.carealert	Samoa	com.jelly.samoa.tracerv2
San Marino	it.ministerodellasalute.coverifica19sm	Saudi Arabia	sa.gov.nic.tawakkalna
Saudi Arabia	co.covid.coronavirus	Saudi Arabia	com.aramco.healthCode
Saudi Arabia	com.tetaman.home	Saudi Arabia	com.lean.sehaty
Saudi Arabia	sa.gov.nic.tabaud	Scotland	scot.nhs.nss.vcert
Scotland	com.spotteron.coronareport	Scotland	gov.scot.covidtracker
SE ASIA	com.trig.tracvirus	Senegal	com.fehudigital.alertesantesn
Seychelles	com.travizory.evisa.sc	Seychelles	com.panafricare.lasante
Singapore	sg.gov.homer	Singapore	sg.gov.hpb.healthhub

TABLE A.3: List of studied COVID-19 Android applications (Part 3/5).

Country	App ID	Country	App ID
Singapore	com.hitachi.shn_mobile	Singapore	sg.gov.mom.sgfwmomcare
Singapore	sg.gov.tech.bluetrace	Singapore	com.contacttracer
Singapore	example.safeentryscanner	Singapore	sg.gov.tech.safeentry
Slovakia	sk.marekgogol.zostanzdravy	Slovakia	sk.it.greenpass
Slovakia	sk.it.overpass	Slovenia	si.nijz.covid
Slovenia	si.gov.ostanizdrav	Slovenia	si.gov.zvem
South Africa	net.epione.reopen.android	South Africa	za.gov.health.covidconnect
South Africa	com.coviid	Spain	org.madrid.ztav.tarjetaSanitariaVirtual
Spain	com.atos.spain.th	Spain	cat.gencat.mobi.StopCovid19Cat
Spain	com.ericetm2m.colabora	Spain	org.madrid.CoronaMadrid
Spain	es.saludinforma.android	Spain	cat.gencat.mobi.confinApp
Spain	es.gob.asistenciacovid19	Spain	es.gva.responde
Spain	es.gva.coronavirus	Spain	com.appandabout.defusing
Spain	com.appfeel.interactiveclinics.dev	Spain	com.mostrarium.sjd
Spain	es.gob.radaracovid	Spain	gal.xunta.covidpass
Spain	es.juntadeandalucia.msspa.saludandalucia	Spain	es.sergas.appbox
Spain	org.gobiernodecanarias.sanidad.scs.micert...	Sri Lanka	com.gatechologies.rekemuapi
Sri Lanka	org.sshield.selfshield	Sri Lanka	app.ceylon.selftrackingapp
Sudan	com.fmoh.sd.covax_reg	Sudan	com.covid.www
Sweden	com.giddir.coronafree	Switzerland	ch.admin.bag.covidcertificate.wallet
Switzerland	ch.covid19bs.app.PMSMobile	Switzerland	com.hug_ge.coronapp
Switzerland	ch.admin.bag.dp3t	Switzerland	com.arit.geohealthapp
Switzerland	ch.admin.bag.covidcertificate.verifier	Taiwan	tw.gov.cdc.exposurenotifications
Taiwan	com.nhiApp.v1	Tanzania	com.rahisi.certificate_scanner
Thailand	com.dga.thailandplus.android	Thailand	com.mor.promplus
Thailand	com.articulus.sydekick	Thailand	th.or.nectec.ddc_care
Thailand	com.devcnx.thaichana_qrcode	Thailand	com.ktb.thaichana.prod
Thailand	com.thaialert.app	Timor Leste	appcovid19naromanxyz.wpapp
Togo	org.caresptogo.covid_19tg	Togo	com.togo.covid19
Trinidad & Tobago	com.mohtt.mohttapp	Tunisia	tn.omne.e7mi
Turkey	tr.gov.saglik.koronaonlem	Turkey	tr.gov.saglik.hayatesvesigar
UAE	com.knasirayaz.mohapcovid	UAE	ae.tracecovid.app
UAE	ae.gov.dha.covid19	UAE	ae.gov.dha.covid19.paramedic
UAE	ae.haad.staysafe.stayathome	UAE	doh.health.shield
Uganda	ug.go.health.mobile.covid19	Uganda	org.definingtechnologies.covidtracer
Uganda	com.ctiafrica.android.accelerator.mohctc	UK	com.instantaccessmedical.mobile.instant...
UK	au.com.vodafone.dreamlabapp	UK	com.joinzoe.covid_zoe
UK	uk.ac.cam.cl.covid19sounds	UK	com.nuasolutions.hse.prod
UK	appinventor.ai_david_taylor.Coronavirus...	UK	com.virustracker.app
UK	com.expertselfcare.covid_19	UK	co.uk.healthcreatives.uclhcovid19
UK	uk.nhs.covid19.production	UK	org.theviralapp.app1
UK	com.eveio.pass	Ukraine	ua.gov.diia.app
Ukraine	ua.gov.diia.quarantine	Uruguay	uy.gub.salud.plancovid19uy
USA	gov.va.mobilehealth.ncptsd.covid	USA	com.clearme.clearapp
USA	uk.co.tracktogether	USA	com.obviohealth.obvio19
USA	jhu.edu.JohnsHopkinsCOVIDControl	USA	org.howwefeel
USA	com.kencorhealth.covid	USA	tracker.healthcare.sentinel
USA	com.carbonhealth.patient.prod	USA	com.openmed.corona
USA	com.patientmpower.covid19.usa	USA	org.cvkey.cvkey
USA	com.stratum.healthcheckguardapp	USA	com.childrensomaha.studentsymptomchecker
USA	com.justmiine.miinehealth	USA	com.campusclear
USA	com.infobeyond.preworkscreen	USA	com.kandasoft.workpass.today
USA	com.sway.clearance19	USA	com.work2live.app.entrsafeapp
USA	edu.cmich.apps.healthscreen	USA	edu.columbia.reopen.cu
USA	edu.uc.covidcheck	USA	edu.umich.ResponsiBLUE
USA	edu.wabash.covidpass	USA	org.usezero.healthpass
USA	com.stratum.healthcheckapp	USA	ai.carespree
USA	com.paxerahealth.CoronaCare	USA	gov.cdc.general
USA	org.covidprotocols.twa	USA	gov.cdc.niosh.PPETracker
USA	edu.mit.privatekit	USA	org.pathcheck.covidsafepaths
USA	mbks	USA	com.currentcareanalytics.coverified
USA	com.virtualmasons.reelhealth	USA	gomeyra.scan
USA	me.safehealth.protectwell	USA - AL	gov.adph.exposurenotifications
USA - AK	gov.ak.covid19.exposurenotifications	USA - AZ	gov.azdhs.covidwatch.android
USA - CA	gov.ca.covid19.exposurenotifications	USA - CO	gov.co.cdph.exposurenotifications
USA - CO	com.soc.mycolorado	USA - CT	gov.ct.covid19.exposurenotifications
USA - DE	gov.de.covidtracker	USA - DC	gov.dc.covid19.exposurenotifications
USA - FL	com.flherm	USA - FL, UT	co.twenty.stop.spread
USA - GA	in.punch.alert	USA - Guam	org.pathcheck.guam.bt
USA - HI	org.alohasafe.alert	USA - HI	org.thecommonsproject.android.phr
USA - IL	edu.illinois.covid	USA - LA	org.pathcheck.la.bt
USA - MD	gov.md.covid19.exposurenotifications	USA - MA	gov.ma.covid19.exposurenotifications
USA - Miami	com.shield.CombatCovidMD	USA - MI	gov.michigan.MiCovidExposure
USA - MN	org.pathcheck.covidsafepathsBt.mn	USA - MS	com.cspire.telehealth
USA - NV	gov.nv.dhhs.en	USA - NJ	com.nj.gov.covidalert
USA - NJ	com.foxhallwythe.docket.mobile	USA - NM	gov.nm.covid19.exposurenotifications
USA - NY	gov.ny.health.proximity	USA - NY	gov.ny.its.healthpassport.verify
USA - NY	gov.ny.its.healthpassport.wallet	USA - NC	gov.nc.dhhs.exposurenotification
USA - OR	gov.or.covid19.exposurenotifications	USA - Palm Beach	com.shield.CombatCovidPBC
USA - PA	gov.pa.covidtracker	USA - Puerto Rico	org.pathcheck.pr.bt
USA - RI	com.ri.crushcovid	USA - Sonoma	com.sococheck

TABLE A.4: List of studied COVID-19 Android applications (Part 4/5).

Country	App ID	Country	App ID
USA – SC	musc.exposurenotification	USA – SD, ND, WY	com.proudcrowd.exposure
USA – TX	com.hornsense	USA – UT	gov.ut.covid19.exposurenotifications
USA – Various	com.expil.novid	USA – VA	gov.vdh.exposurenotification
USA – WA	gov.wa.doh.exposurenotifications	USA – WI	gov.wi.covid19.exposurenotifications
USA – WY	com.proudcrowd.care	Uzbekistan	uz.uzinfocom.selfsafety
Uzbekistan	uz.uicdevelopment.birgayengamiz	Vietnam	com.covidtrack
Vietnam	vn.coquan.hd	Vietnam	vn.dtt.imsafe
Vietnam	one.galaxy.govn	Vietnam	com.family.tokhaiyte
Vietnam	com.vnptit.innovation.ncovi	Vietnam	gov.moh.antoancovid
Vietnam	com.Eha.covid_19	Vietnam	net.cungmua365.ncovigialai
Vietnam	vn.suckhoetoandan.v2	Vietnam	com.mic.bluezone
Zimbabwe	com.dencroft.covidapp		

TABLE A.5: List of studied COVID-19 Android applications (Part 5/5).

Appendix B

Detailed Information of Other Analyzed Trackers

In this Appendix, we complete the detailed information of analyzed trackers. The most remarkable trackers were already discussed in Chapter 4.

B.1 AdColony

B.1.1 Introduction

Founded in 2008, AdColony (purchased in 2021 by Digital Turbine) is a mobile video advertising company and features its proprietary Instant-Play technology that serves full-screen video ads instantly in HD across its network of iOS and Android apps. Its website www.adcolony.com redirects to its parent company site, www.digitalturbine.com.

It has been involved in previous privacy issues, for example, in 2021 the Norwegian Consumer Council filed a complaint against it since it collected data from app *Grindr* without the proper consent of users (<https://www.datatilsynet.no/en/news/2021/intention-to-issue--10-million-fine-to-grindr-llc2/>).

B.1.2 Tracked Information

Table B.1 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Mobile Advertising
Push Notifications	-
AAID	Android id Limit tracking
User ID	Session id Zone ids
Location SW	Country locale Country locale short Language Timezone ietf Timezone gmt Timezone dst
Location HW	-
Device SW	OS version release Platform / OS name ("android") Version SDK / Device API OS arch
Device HW	Device manufacturer Device model Type (tablet, phone)
APK	Package name Package version name Package version code Appid Environment App bundle name App bundle version Dark mode Available stores Permissions
Applications and Processes	-
Disk and Memory	Memory class Memory used
Network	Carrier Network type MAC address Network speed Cell service country code Clear text permitted
Screen and Audio	Display width Display height Screen width Screen height DPI Current orientation Density
Rooted, Jailbroken, Emulated and Simulated	-
Time	Date (of error)
Battery	Battery info Battery level
SDK	SDK version Controller version SDK type
Others	Index Level (of error) Message (log) Data path Media path Temp storage path Launch metadata

TABLE B.1: Tracked information by AdColony.

B.1.3 Data Flow Diagram

It can be seen that AdColony gathers the information from two main points located in classes *C0968ak* and *C0856a*. In the former, when method *m15378c* is fired, it gets the information from the mentioned class, and then other methods are called in a chain until the gathered information is finally sent to its server. For the latter case, the process is completed in two stages, where in the first part the data is collected and stored in a field from class *C1068t*, and then this field is queried to send the information at the end to AdColony's servers.

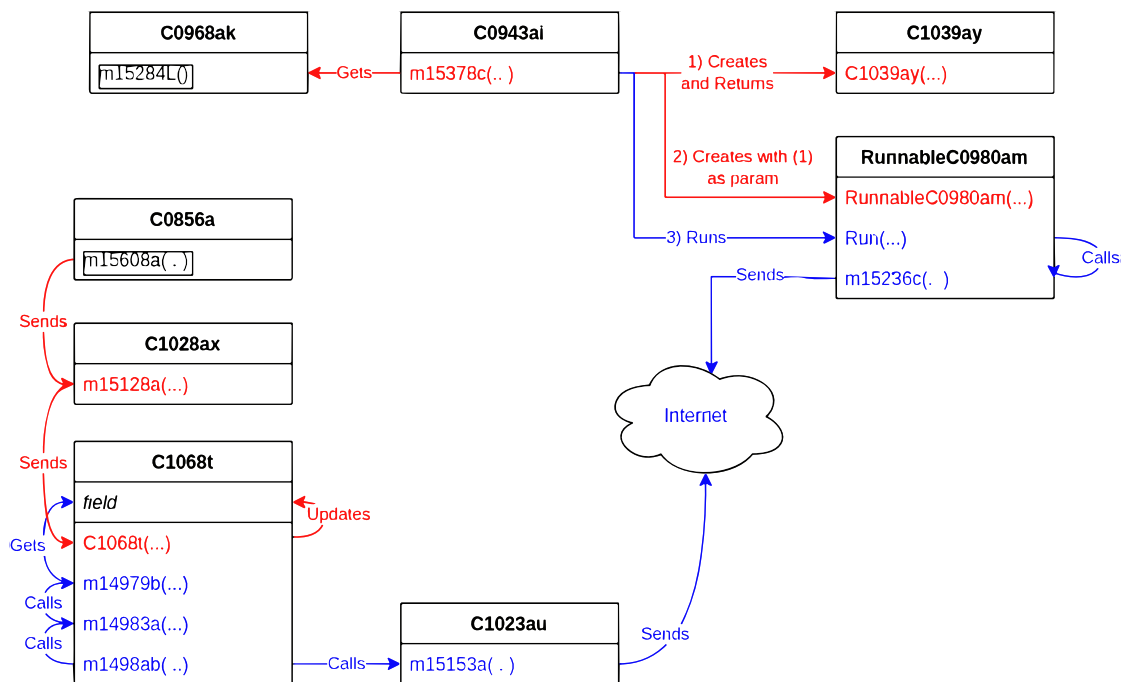


FIGURE B.1: Data flow in AdColony

B.2 AltBeacon

B.2.1 Introduction

AltBeacon allows Android devices to use beacons like iOS devices do. An app can request to get notifications when one or more beacons appear or disappear. An app can also request to get a ranging update from one or more beacons at a frequency of approximately 1Hz. It is an open source project (<https://github.com/AltBeacon/android-beacon-library>) lead by radiusnetworks.com. Its documentation can be found at <https://altbeacon.github.io/android-beacon-library/>.

The authors created this project motivated by the lack of an open and interoperable specification for proximity beacons (<https://altbeacon.org/>).

B.2.2 Tracked Information

No data was found to be tracked by AltBeacon.

B.2.3 Data Flow Diagram

Since no information was detected to be tracked, there is no data flow.

B.3 AppNext

B.3.1 Introduction

Created in 2012, AppNext's platform provides mobile publishers and app marketers with end-to-end technology solutions for premium monetization and app growth. They state that they have created a marketplace that connects publishers and advertisers directly and transparently, amplifying their monetization and advertising efforts. It recommends apps to users "based on their needs". Purchased by Affle company, its website can be found at www.appnext.com.

B.3.2 Tracked Information

Table [B.2](#) details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	-
AAID	-
User ID	Session id ID CUID
Location SW	Timezone Language Locality Country code
Location HW	Latitude Longitude Location Time Activity transition
Device SW	OS version SDK OS version release DEvn ("android")
Device HW	Device manufacturer Device model Device brand Device name
APK	Package id / Name / App package Duse / Non-market apps
Applications and Processes	Activities running DefLun InsLun Number of activities Installed apps
Disk and Memory	Filesystem access
Network	Network type Carrier Wifi SSID Near SSIDs Bluetooth activated Phone operator Flight mode Name of Bluetooth connected
Screen and Audio	Screen brightness DPI Resolution Font scale Ringer mode Earplug connected Device volume Orientation Is screen on
Rooted, Jailbroken, Emulated and Simulated	-
Time	Client date
Battery	Device charging
SDK	-
Others	Ad TID Ad VID AUID Did privacy (false) G (contacts) OSID Camera activated in last 15 mins SSIDs by location Data map

TABLE B.2: Tracked information by AppNext.

B.3.3 Data Flow Diagram

From four main sources, AppNext collects data. The method `mo14119a(...)` is implemented by several classes. This method collects data that later is sent to their servers in class `C1374f`. This class also collects information by itself in the method `m14311e(...)`, and it is called by class `C1306k` which harvests data as well. Finally, in class `C1321utils` data is also gathered and, after several calls in chain, sent to the servers.

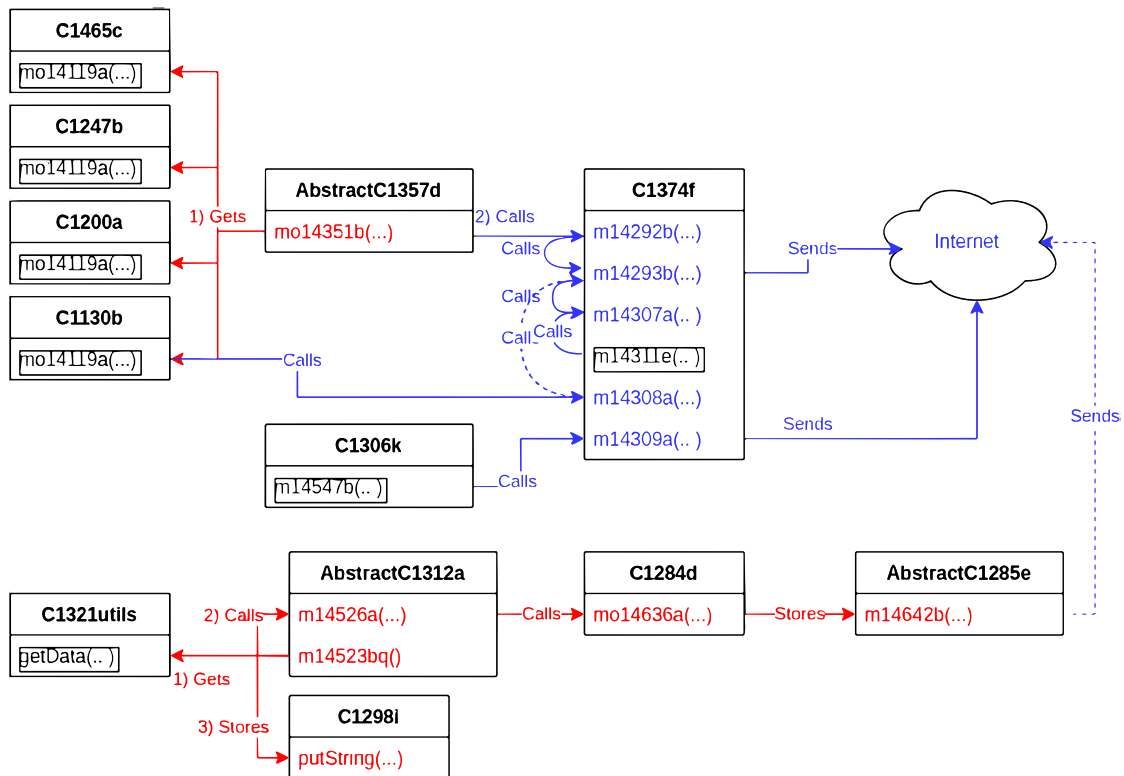


FIGURE B.2: Data flow in AppNext

B.4 Braze

B.4.1 Introduction

Braze (previously known as Appboy) is a customer engagement platform used by businesses for multichannel marketing. It was founded in 2011. It is considered one of the leaders in mobile marketing and customer engagement platforms. Its website is www.braze.com.

B.4.2 Tracked Information

Table B.3 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	Yes
AAID	Ad ID Ad tracking enabled
User ID	Device id (UUID) User id
Location SW	Locale Timezone
Location HW	Event location Event geofence
Device SW	OS version
Device HW	Device model
APK	App version App version code
Applications and Processes	-
Disk and Memory	-
Network	Carrier
Screen and Audio	Height Width
Rooted, Jailbroken, Emulated and Simulated	-
Time	Timestamp (event) Time (event) Last time sync Last card updated
Battery	-
SDK	SDK flavor SDK version
Others	Background restricted Feed Triggers Config Trigger id Trigger event type Data Device logs

TABLE B.3: Tracked information by Braze.

B.4.3 Data Flow Diagram

As in other cases, in Braze it can be appreciated that several classes implement the method `mo564365(...)`. These are called from class `RunnableC0814de` and then sent to the servers. Similarly, two classes implement the method `mo56091b(...)`, where data is collected and sent to internet as in the previously described process.

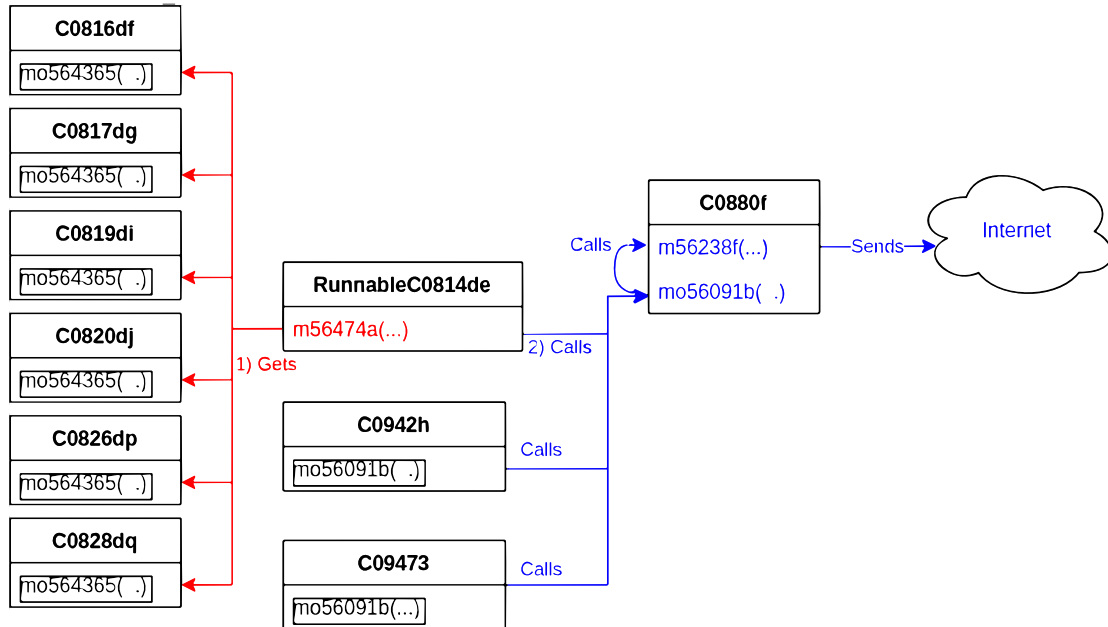


FIGURE B.3: Data flow in Braze

B.5 Google Firebase Analytics

B.5.1 Introduction

As a complete development and backend framework, it offers a myriad of products and services like: Cloud Firestore, Extensions, App Check, Cloud Functions, Authentication, Hosting, Cloud Storage, Realtime Database, Crashlytics, Performance Monitoring, Test Lab, App Distribution, Google Analytics, Machine Learning, In-App Messaging, A/B Testing, Cloud Messaging, Remote Config, and Dynamic Links. Hence, it is not a surprise that it was the most used third-party library in the study. Furthermore, it can be integrated with several other solutions, like: Google Ads, AdMob, Google Marketing Platform, Google Play, Data Studio, BigQuery, Slack, Jira, and PagerDuty.

Launched in 2011, it was acquired by Google in 2014. As an example of its relevance in the current Android ecosystem, Firebase superseded Google Cloud Messaging service for push notifications.

A privacy lawsuit accused Google “of violating federal wiretap law and California privacy law by logging what users are looking at in news, ride-hailing and other types of apps despite them having turned off “Web & App Activity” tracking in their Google account settings.”. “...the data collection happens through Google’s Firebase, a set of software popular among app

Category	Details
Tracker Category	Development and Backend Framework
Push Notifications	Yes
AAID	-
User ID	Firebase installation id
Location SW	-
Location HW	-
Device SW	OS version SDK
Device HW	Device product Device device Device brand
APK	App id X-Android-Package
Applications and Processes	-
Disk and Memory	-
Network	-
Screen and Audio	-
Rooted, Jailbroken, Emulated and Simulated	-
Time	-
Battery	-
SDK	Auth version Project identifier
Others	IID token Fire core Kotlin

TABLE B.4: Tracked information by Google Firebase Analytics.

makers for storing data, delivering notifications and ads, and tracking glitches and clicks. Firebase typically operates inside apps invisibly to consumers.” (<https://www.reuters.com/article/us-alphabet-google-privacy-lawsuit-idUSKCN24F2N4>). The lawsuit was later dismissed in court (https://app.ediscoveryassistant.com/case_law/41662-rodriquez-v-google-llc).

B.5.2 Tracked Information

Table B.4 details the information tracked by this tracker (up to ten elements per category).

B.5.3 Data Flow Diagram

For this tracker, data is collected in class *FirebaseCommonRegistrar* method *getComponents(...)*. This method updates a field from class *LibraryVersion*, which later is gotten by method *getUserAgent(...)* from class *DefaultUserAgentPublisher* and finally sent in method *openHTTPURLConnection(...)* from class *FirebaseInstallationServiceClient*.

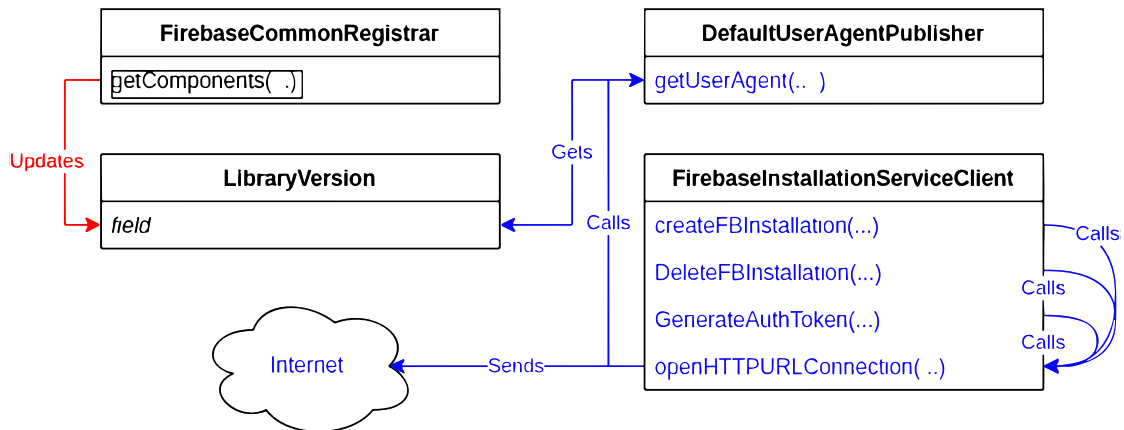


FIGURE B.4: Data flow in Google Firebase Analytics

B.6 Google Tag Manager

B.6.1 Introduction

As its name implies, it is a tag management system provided by Google to manage JavaScript and HTML tags, including web beacons for web tracking and analytics. It was released in 2012. Simply put, it allows developers to manage and deploy marketing tags (snippets of code or tracking pixels) on their website or mobile apps without having to change their code. The data collected with the tags can be later used as input in analytics platforms. The tool can be accessed at <https://tagmanager.google.com/>.

It has been used by malicious parties, especially recently, as a weapon to inject malicious code within commercial (<https://geminiadvisory.io/magecart-google-tag-manager/>). Hence, caution is advised when using this tool.

B.6.2 Tracked Information

No data was found to be tracked by Google Tag Manager.

B.6.3 Data Flow Diagram

Since no information was detected to be tracked, there is no data flow.

Category	Details
Tracker Category	Crash Reports and Monitoring
Push Notifications	-
AAID	-
User ID	Device id
Location SW	Address / Country Code
Location HW	-
Device SW	Application framework version Agent name OS build OS name OS version OS arch
Device HW	Device manufacturer Device model
APK	App framework App name App version App build Package id
Applications and Processes	Thread id Thread name Runtime (jvm version)
Disk and Memory	Disk available Memory usage
Network	Network status Network WAN Type
Screen and Audio	Orientation
Rooted, Jailbroken, Emulated and Simulated	-
Time	-
Battery	-
SDK	Version
Others	Misc

TABLE B.5: Tracked information by New Relic.

B.7 New Relic

B.7.1 Introduction

Founded in 2008, New Relic provides app performance and troubleshooting management services, under several products it offers: New Relic APM, New Relic Mobile, New Relic Browser, New Relic Synthetics, New Relic Servers and New Relic Insights. Its website is <https://newrelic.com/>.

B.7.2 Tracked Information

Table B.5 details the information tracked by this tracker (up to ten elements per category).

B.7.3 Data Flow Diagram

In New Relic, the data tracking process is quite simple. Device information is collected in class *AndroidAgentImpl*, which, after several classes calling, is stored in a field in class *HarvestData* (another quite revealing class name). At the moment of sending the collected data to their servers, this field and another field containing connection information are queried.

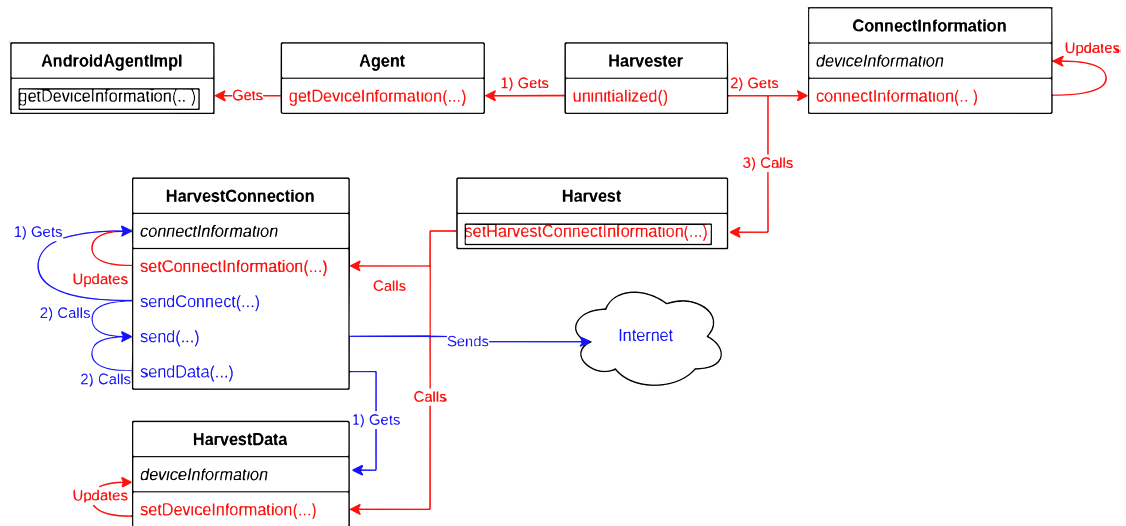


FIGURE B.5: Data flow in New Relic

B.8 Open Telemetry

B.8.1 Introduction

Different to other third-party libraries studied, Open Telemetry is a collection of tools, APIs, and SDKs used to instrument, generate, collect, and export telemetry data (metrics, logs, and traces), to help with analyzing app's performance and behavior. In other words, it is an observability framework. It is free and open-source, formed after the merge of Open Tracing and Open Census in 2019. Currently, it is a project from CNCF, and the second most active after Kubernetes. It can be found at <https://opentelemetry.io/>.

B.8.2 Tracked Information

No data was found to be tracked by Google Tag Manager.

B.8.3 Data Flow Diagram

Since no information was detected to be tracked, there is no data flow.

Category	Details
Tracker Category	Push Notification
Push Notifications	Yes
AAID	-
User ID	HW id
Location SW	Timezone Language
Location HW	-
Device SW	OS version Device type
Device HW	Device model Device manufacturer Device name
APK	Package name App version
Applications and Processes	-
Disk and Memory	-
Network	SIM operator Connection type Connection type name Connection sub type Connection sub type name
Screen and Audio	-
Rooted, Jailbroken, Emulated and Simulated	Jailbroken
Time	Timestamp UTC Timestamp current
Battery	-
SDK	Version
Others	Application Notification Type Sounds Push stat metadata Message delivery metadata Features Action

TABLE B.6: Tracked information by Pushwoosh.

B.9 Pushwoosh

B.9.1 Introduction

As its name may indicate, Pushwoosh provides engagement services related to push notifications. Using their SDK, developers can segment, communicate, experiment, engage, convert and retain mobile app users and website visitors. Founded in 2014, its website is www.pushwoosh.com.

Recently there have been some controversies, since a report from Reuters stated that Pushwoosh origins are linked to Russia and its SDK was found in apps like the Center for Disease Control and Prevention from the USA government or the USA Army app.

B.9.2 Tracked Information

Table B.6 details the information tracked by this tracker (up to ten elements per category).

B.9.3 Data Flow Diagram

Pushwoosh harvesting of data is very simple. Class *PushRequest* is in charge of collecting the data and immediately calling class *C232b*, which sends the harvested information to the servers.

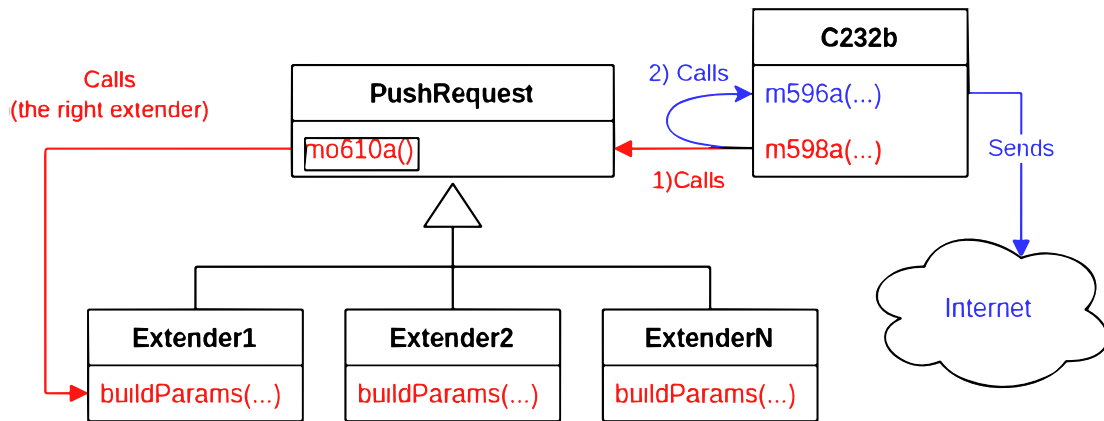


FIGURE B.6: Data flow in Pushwoosh

B.10 Segment

B.10.1 Introduction

Founded in 2011, Segment provides app usage, engagement and audience services. For that, it collects data from mobile apps and helps developers take informed decisions regarding their users. In 2020, it was acquired by Twilio, another customer engagement company. Its website is <https://segment.com/>.

B.10.2 Tracked Information

Table B.7 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	-
AAID	Ad id Limit ad tracking
User ID	Id
Location SW	Timezone Locale
Location HW	-
Device SW	User agent Name Version
Device HW	Device model Device manufacturer Device name Device type
APK	Name Version Namespace Build
Applications and Processes	-
Disk and Memory	-
Network	Wifi Bluetooth Cellular Carrier
Screen and Audio	Density Height Width
Rooted, Jailbroken, Emulated and Simulated	-
Time	-
Battery	-
SDK	Version Name
Others	Events

TABLE B.7: Tracked information by Segment.

B.10.3 Data Flow Diagram

The class *Analytics* from Segment is in charge of collecting the information, with the assistance of classes *Client* and *ConnectionFactory*. Following collecting the data, it sends the data to its servers.

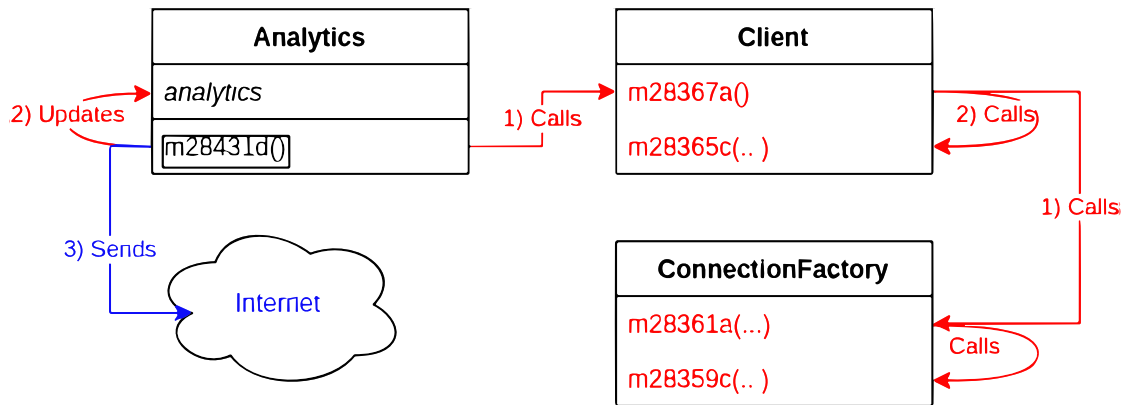


FIGURE B.7: Data flow in Segment

B.11 Splunk MINT

B.11.1 Introduction

MINT by Splunk was a library used for real time crash and error reporting, event analysis and monitoring. It also provided other features as engagement, screen tracking, and user retention. It was discontinued at the end of 2021.

B.11.2 Tracked Information

Table B.8 details the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	Crash Reports and Monitoring
Push Notifications	-
AAID	-
User ID	UUID Username Session id
Location SW	Locale
Location HW	-
Device SW	Platform Os version
Device HW	Device model Device manufacturer
APK	Package name App version App version code Environment ("release")
Applications and Processes	-
Disk and Memory	Filesystem encrypted Mem sys low Mem sys total Mem sys available Mem sys threshold Mem app max Mem app available Mem app total
Network	Carrier Connection type Connection subtype Connection state GPS status
Screen and Audio	Current view Orientation
Rooted, Jailbroken, Emulated and Simulated	Rooted
Time	Milliseconds from start Session duration
Battery	Battery level
SDK	SDK version
Others	Extra data Custom data Transactions Event name Level (Verbose, Debug, Info, Warning, Error) Stacktrace Breadcrumbs Log

TABLE B.8: Tracked information by Splunk MINT.

B.11.3 Data Flow Diagram

In Splunk MINT, data is collected in classes *BaseDTO* and *ActionX*. After its collection, the data is sent to the servers from class *NetSender*.

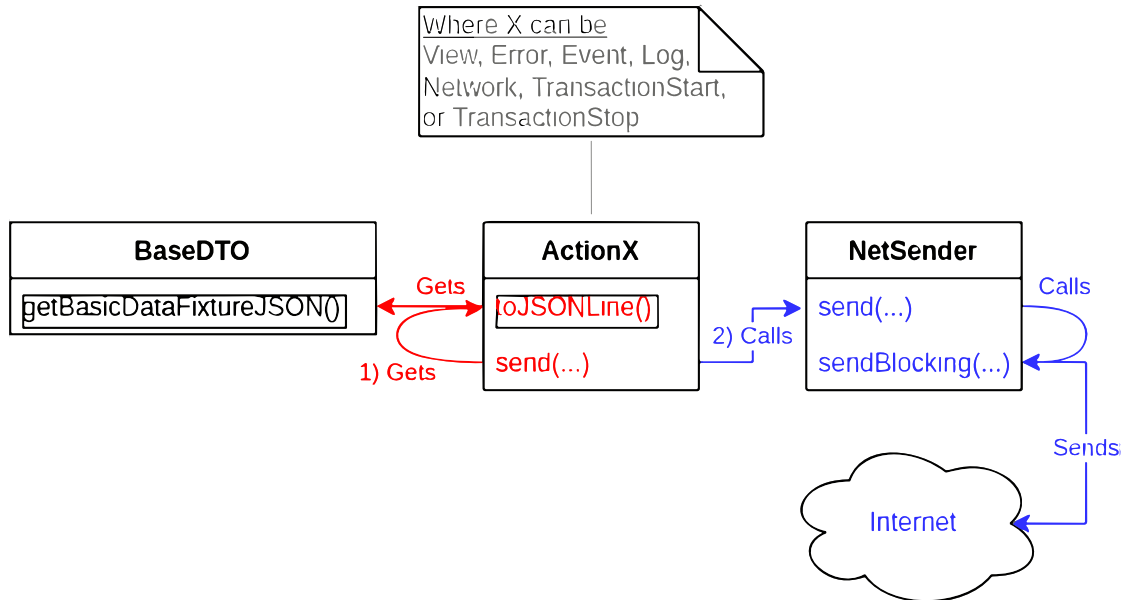


FIGURE B.8: Data flow in Splunk MINT

B.12 Startapp

B.12.1 Introduction

Recently rebranded as Start.io, it is a mobile marketing and audience platform library founded in 2011. Its website is www.start.io.

B.12.2 Tracked Information

Tables B.9 and B.10 detail the information tracked by this tracker (up to ten elements per category).

Category	Details
Tracker Category	App Usage, Audience and Engagement
Push Notifications	-
AAID	User Advertising Id Limited tracking Advertising id source ("APP")
User ID	Ltr id User id Client session id
Location SW	Locale Locale list Input languages
Location HW	Accuracy Vertical accuracy Altitude Bearing Latitude Longitude Speed Time Provider Timestamp (1 more element)
Device SW	OS OS version release OS system version OS version SDK
Device HW	Device model Device manufacturer build fingerprint Sim operator Sim operator name TAC / Manufacturer code Sim state Phone count Phone type Sim infos (9 more elements)
APK	App target version Product id Package id Installer package Outsource App version name App version code Is ddbg
Applications and Processes	Foreground app
Disk and Memory	Memory free Memory total Memory used Memory state Used ram Free ram
Network	Cell identity longitude Roaming Signal Level Preferred Network Flight mode Mobile data enabled Connection type Operator name Service state Bluetooth (102 more elements)
Screen and Audio	Screen on / Screen state Orientation Width Height Density
Rooted, Jailbroken, Emulated and Simulated	Is rooted Root Simulator
Time	Location age Device drift millis Millis since last sync Device up time / Time since boot (elapsed real time) Duration overall no sleep Duration Overall Timestamp (event) Timestamp Date Timestamp Offset Timestamp Millis (1 more element)

TABLE B.9: Tracked information by Startapp. (Part 1/2)

Category	Details
Battery	Battery capacity Battery current Battery remaining energy Battery voltage Battery status Battery health Battery temperature Battery charge plug Battery technology Battery level (2 more elements)
SDK	SDK version Flavor Frameworks data SDK id
Others	Measurement Type IsSynced Is Default Voice Sim Is Default Data Sim Airport code Category (event) Value (event) App activity (stack trace) Wifi info Call State (24 more elements)

TABLE B.10: Tracked information by Startapp. (Part 2/2)

B.12.3 Data Flow Diagram

Most of the harvested data by Startapp is gathered in class *C7380b*, although in classes *C7299e*, *C7294a* and *AbstractC7199c* some collection is done as well. All the tracked information is later sent to the servers from class *C6858e*.

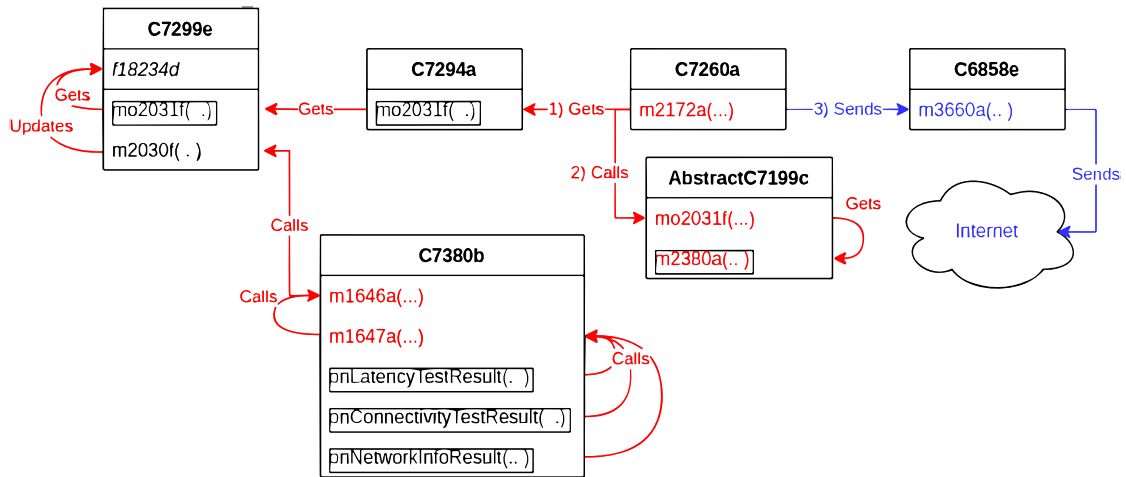


FIGURE B.9: Data flow in Startapp

Appendix C

Examples of Trackers Code

In this Appendix, we provide code examples of how trackers collect information about the application, the device, and use, as seen in their code.

Android Advertisement ID (AAID):

```
adid = Settings.Secure.getString(contentResolver, "advertising_id");
```

Or using reflection:

```
Object invoke = Class.forName("com.google.android.gms.ads.identifier.  
    AdvertisingIdClient").getMethod("getAdvertisingIdInfo", Context.class).  
    invoke(null, context);  
adid = invoke.getClass().getMethod("getId", new Class[0]).invoke(invoke, new  
    Object[0]);
```

User ID::

```
user_id = UUID.randomUUID().toString();
```

Location Software:

```
country_locale = Locale.getDefault().getCountry();  
language = Locale.getDefault().getLanguage();  
timezone = TimeZone.getDefault();  
daylight_saving = Calendar.getInstance().getTimeZone().inDaylightTime(new Date  
    ());
```

Location Hardware:

```
location_latitude = location.getLatitude();  
location_longitude = location.getLongitude();  
location_accuracy = location.getAccuracy();  
location_provider = location.getProvider();  
location_vertical_accuracy = location.getVerticalAccuracyMeters();  
location_altitude = location.getAltitude();  
location_bearing = location.getBearing();  
location_speed = location.getSpeed();
```

```
location_time = location.getTime();
```

Besides, most of the trackers would check if they had the right permissions to access the device location, checking as follows:

```
if (context.getPackageManager().checkPermission("android.permission.
    ACCESS_COARSE_LOCATION"...
...
if (context.getPackageManager().checkPermission("android.permission.
    ACCESS_FINE_LOCATION"...
```

Or:

```
if (context.checkCallingOrSelfPermission("android.permission.
    ACCESS_COARSE_LOCATION")...
...
if (context.checkCallingOrSelfPermission("android.permission.
    ACCESS_FINE_LOCATION")...
```

Device Software:

```
os_version1 = Build.VERSION.RELEASE;
os_version2 = Build.VERSION.SDK_INT;
System.getProperty("os.arch").toLowerCase(Locale.ENGLISH);
user_agent1 = WebSettings.getDefaultUserAgent(context);
user_agent1 = System.getProperty("http.agent");
```

Device Hardware:

```
manufacturer = Build.MANUFACTURER;
model = Build.MODEL;
display = Build.DISPLAY;
...
try {
    Class.forName("com.amazon.device.messaging.ADM");
    device_type = "Amazon";
} catch (ClassNotFoundException unused) {
    device_type = "Android";
}
...
TelephonyManager telephonyManager = context.getSystemService("phone");
sim_operator = telephonyManager.getSimOperator();
sim_operator_name = telephonyManager.getSimOperatorName();
sim_state = telephonyManager.getSimState();
...
SubscriptionManager x = SubscriptionManager.from(context);
active_sim_count = x.getActiveSubscriptionInfoCount();
```

Android Package Kit (APK):

```

app_name = context.getPackageName();
app_version = context.getPackageManager().getPackageInfo(context.getPackageName
    (), 0).versionName;
app_build_number = context.getPackageManager().getPackageInfo(context.
    getPackageName(), 0).versionCode;
...
PackageManager packageManager = context.getApplication().getPackageManager();
app_bundle_name = packageManager.getApplicationLabel(packageManager.
    getApplicationInfo(context.getPackageName(), 0));
app_bundle_version = context.getPackageManager().getPackageInfo(context.
    getPackageName(), 0).versionName;
...
PackageInfo packageInfo = context.getPackageManager().getPackageInfo(context.
    getPackageName(), 4096);
JSONArray requested_permissions = new JSONArray();
for (int i = 0; i < packageInfo.requestedPermissions.length; i++)
    requested_permissions.put(packageInfo.requestedPermissions[i]);
...
String nameForUid = context.getPackageManager().getNameForUid(Binder.
    getCallingUid())
is_instant_app = context.getPackageManager().isInstantApp(nameForUid);

```

Applications/Processes:

```

List<ActivityManager.RunningTaskInfo> runningTasks;
ArrayList activities_running = new ArrayList();
UsageStatsManager usageStatsManager = context.getSystemService("usagestats");
currentTimeMillis = System.currentTimeMillis();
List<UsageStats> queryUsageStats = usageStatsManager.queryUsageStats(4,
    currentTimeMillis - j, currentTimeMillis);
ListIterator<UsageStats> listIterator = queryUsageStats.listIterator();
while (listIterator.hasNext()) {
    UsageStats next = listIterator.next();
    if (!usageStatsManager.isAppInactive(next.getPackageName()) && next.
        getTotalTimeInForeground() >= j2 && !str.contains("com.android"))
        activities_running.add(next.getPackageName());
}
...
PackageManager packageManager = context.getPackageManager();
Intent intent = new Intent("android.intent.action.MAIN", (Uri) null);
intent.addCategory("android.intent.category.LAUNCHER");
number_of_activities = packageManager.queryIntentActivities(intent, 0).size();
...
for (ApplicationInfo applicationInfo : packageManager.getInstalledApplications
    (128))
    if (applicationInfo != null && (applicationInfo.flags & 1) == 0)

```

```

        installed_apps.add(applicationInfo);
    ...
int myUid = Process.myUid();
is_privileged_process = myUid == 0 || myUid == 1000;
...
ActivityManager activityManager = context.getSystemService("activity");
app_state = BACKGROUND;
for (ActivityManager.AppTask appTask : activityManager.getAppTasks())
    if (appTask.getTaskInfo().isRunning)
        app_state = FOREGROUND;
...
thread_id = Thread.currentThread().getId();
thread_name = Thread.currentThread().getName();
...
jvm_version = System.getProperty("java.vm.version");

```

Disk/Memory:

```

ActivityManager activityManager = context.getSystemService("activity");
memory_class = activityManager.getMemoryClass();
Runtime runtime = Runtime.getRuntime();
memory_usage = runtime.totalMemory() - runtime.freeMemory();
max_memory = runtime.maxMemory();
...
filesystem_access = (context.checkPermission("android.permission.
    READ_EXTERNAL_STORAGE", Process.myPid(), Process.myUid()) == 0);
filesystem_encrypted = (3 == (context.getSystemService("device_policy").
    getStorageEncryptionStatus()));
...
disk_free = Environment.getDataDirectory().getUsableSpace();
long j = statFs.getBlockSizeLong();
long j2 = statFs.getAvailableBlocksLong();
long j3 = statFs.getBlockCountLong();
disk_size_free = j2 * j
disk_size_total = j3 * j

```

Network:

```

TelephonyManager telephonyManager = context.getSystemService("phone");
carrier = telephonyManager.getNetworkOperatorName();
...
ConnectivityManager connectivityManager = context.getApplicationContext().
    getSystemService("connectivity");
NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
network_type = activeNetworkInfo.getType();
network_type_name = activeNetworkInfo.getTypeName();
network_subtype_name = activeNetworkInfo.getSubtypeName();

```

```

...
ssid = (context.getApplicationContext().getSystemService("wifi")).
    getConnectionInfo().getSSID();
...
BluetoothAdapter defaultAdapter = BluetoothAdapter.getDefaultAdapter();
bluetooth_activated = defaultAdapter.isEnabled();
ble = context.getPackageManager().hasSystemFeature("android.hardware.
    bluetooth_le");
...
ip_address = InetAddress.getHostAddress();
...
gps_enabled = (context.getSystemService("location")).isProviderEnabled("gps");

```

Screen/Audio:

```

DisplayMetrics displayMetrics = new DisplayMetrics();
WindowManager windowManager = context.getSystemService("window");
windowManager.getDefaultDisplay().getMetrics(displayMetrics);
width = displayMetrics.widthPixels;
height = displayMetrics.heightPixels;
dpi = displayMetrics.densityDpi;
density = context.getResources().getDisplayMetrics().density;
screen_brightness = Settings.System.getInt(context.getContentResolver(), "
    screen_brightness");
PowerManager powerManager = context.getSystemService("power");
screen_on = (Build.VERSION.SDK_INT >= 20 && powerManager.isInteractive()) || (
    Build.VERSION.SDK_INT < 20 && powerManager.isScreenOn());
screen_orientation = context.getApplicationContext().getResources().
    getConfiguration().orientation;
...
AudioManager audioManager = context.getSystemService("audio");
ringer_mode = audioManager.getRingerMode();
earplug_connected = audioManager.isWiredHeadsetOn();
volume = return audioManager.getStreamVolume(stream);
music_active = audioManager.isMusicActive();
speaker_on = audioManager.isSpeakerphoneOn();

```

Rooted/Jailbroken/Emulated/Simulated:

```

jailbroken = !string_is_empty(context.getPackageManager().
    getInstallerPackageName(context.getPackageName()));
ROOT_INDICATORS = {"/system/xbin/su", "/system/bin/su", "/system/app/Superuser.
    apk", "/system/app/SuperSU.apk", "/system/app/Superuser", "/system/app/
    SuperSU", "/system/xbin/daemonsu", "/su/bin"};
for (String str : ROOT_INDICATORS) {
    if (!new File(str).exists()) {
...

```

```

emulated = MoreExecutors.startsWith$default(Build.FINGERPRINT, "unknown", false
, 2) || StringNumberConversions.contains$default(Build.FINGERPRINT, "
generic", false, 2) || StringNumberConversions.contains$default(Build.
FINGERPRINT, "vbox", false, 2);
simulated = Build.DEVICE.startsWith("generic");

```

Time:

```

event_timestamp = new Date(System.currentTimeMillis());
...
startTimeMs = SystemClock.elapsedRealtime(); //created at a given time
Long.valueOf(SystemClock.elapsedRealtime() - startTimeMs);
...
install_time context.getPackageManager().getPackageInfo(context.getPackageName
(), 0).firstInstallTime;
update_time context.getPackageManager().getPackageInfo(context.getPackageName()
, 0).lastUpdateTime;

```

Battery:

```

Intent registerReceiver = context.registerReceiver(null, new IntentFilter("
android.intent.action.BATTERY_CHANGED"));
battery_level = registerReceiver.getIntExtra("level", -1) / registerReceiver.
getIntExtra("scale", -1);
battery_charging = registerReceiver.getIntExtra("plugged", -1);
battery_charging2 = registerReceiver.getIntExtra("status", -1);
battery_temperature = registerReceiver.getIntExtra("temperature", -1);
battery_health = registerReceiver.getIntExtra("health", -1);
battery_voltage = registerReceiver.getIntExtra("voltage", -1);
battery_technology = registerReceiver.getStringExtra("technology");
...
BatteryManager batterymanager = context.getSystemService("batterymanager");
batteryInfo[n] = batterymanager.getIntProperty(n); //iterating over n different
properties

```
