How Does Misconfiguration of Analytic Services Compromise Mobile Privacy?

Xueling Zhang
University of Texas at San Antonio
San Antonio, TX, USA
xueling.zhang@utsa.edu

Xiaoyin Wang
University of Texas at San Antonio
San Antonio, TX, USA
xiaoyin.wang@utsa.edu

Rocky Slavin
University of Texas at San Antonio
San Antonio, TX, USA
rocky.slavin@utsa.edu

Travis Breaux
Carnegie Mellon University
Pittsburgh, PA, USA
breaux@cs.cmu.edu

Jianwei Niu
University of Texas at San Antonio
San Antonio, TX, USA
jianwei.niu@utsa.edu

ABSTRACT

Mobile application (app) developers commonly utilize analytic services to analyze their app users' behavior to support debugging, improve service quality, and facilitate advertising. Anonymization and aggregation can reduce the sensitivity of such behavioral data, therefore analytic services often encourage the use of such protections. However, these protections are not directly enforced so it is possible for developers to misconfigure the analytic services and expose personal information, which may cause greater privacy risks. Since people use apps in many aspects of their daily lives, such misconfigurations may lead to the leaking of sensitive personal information such as a users' real-time location, health data, or dating preferences. To study this issue and identify potential privacy risks due to such misconfigurations, we developed a semi-automated approach, Privacy-Aware Analytics Misconfiguration Detector (PAMDroid), which enables our empirical study on misconfigurations of analytic services. This paper describes a study of 1,000 popular apps using top analytic services in which we found misconfigurations in 120 apps. In 52 of the 120 apps, misconfigurations lead to a violation of either the analytic service providers' terms of service or the app's own privacy policy.

KEYWORDS

Privacy, Mobile Application, Program Analysis, Analytic Services, Configuration

ACM Reference Format:
Xueling Zhang, Xiaoyin Wang, Rocky Slavin, Travis Breaux, and Jianwei Niu. 2020. How Does Misconfiguration of Analytic Services Compromise Mobile Privacy?. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380401

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380401

1 INTRODUCTION

Mobile apps often rely on third-party services to enhance user experience through features such as social network integration and crash analysis. Among the most popular types of third-party services, *analytic services* enable app developers to gather user behavior information to improve their products and monetize their apps with targeted ads. Such analytic services can be integrated into apps through package libraries to collect user activities and send user behavior to their servers for analysis. Server-side analysis can then generate aggregated reports for the app's developers. For example, such aggregated reports may describe how many users are from New York City, how many users reached a specific activity, or how long they tend to spend on a specific activity.

Analytic services provide specific methods that allow app developers to set attributes for their users, we refer to those methods as *Attributes Setting Methods (ASMs)*. For example, one commonly used category of ASMs is "set user identifier", which allows app developers to store a user ID for the individual using their apps. These methods are usually optional and can be used to recognize the same user across multiple usages of an app. Once a unique ID is assigned through such a method, the user's behavior reports will be labeled with the provided user ID. These identifiers are strictly used for identification with respect to the service and do not need to be *personally* identifying. For example, a random, unique number or hash value could be used instead of an email address. Using personally identifiable information (PII)[1] as an ID would be considered as bad practice in this case as it presents an unnecessary exposure of sensitive data. By misusing PII (e.g., email, username, device ID) with ASMs this effectively un-anonymizes the reports produced by the analytics service resulting in privacy risk. Furthermore, such misuse may violate the app's own privacy policy, the analytic service providers' terms of service, or general best practices (e.g., data overuse, least privilege).

A major privacy risk associated with third-party analytic services is the data usage after the behavioral reports have been collected by the analytic service. Once the data have left the app and reached an analytic service, the developers and users lose control of the information. Even if the third-party service is trusted not to misuse the data, accumulated long-term storage of un-anonymized user

[1] We use the union of GDPR and Google Analytics definitions for PII[14, 31].

behavioral data is susceptible to theft or leakage [1, 6, 17]. Not expecting PII to exist in the collected behavioral reports, third-party services may share the data to their business partners or do not provide enough protection for them. Furthermore, when multiple apps use the same PII for the same analytic service, multiple behavioral reports can be combined to build more comprehensive personal profiles.

Legal requirements such as EU General Data Protection Regulation (GDPR) requires lawful basis (e.g. legal obligation, explicit consent) to process users' data [15], unless the data is anonymized[13]. For these reasons, it is imperative that unnecessary use of PII for behavioral-report labeling to be eliminated.

Many of the most commonly used analytic services provide documentation specifically discouraging or prohibiting the use of PII as user attributes when using their ASMs. For example, Google's Firebase [3] includes the following in their documentation [10] for configuration of ASM `setUserProperty()`:

> *"When you set user properties, be sure to never include personally identifiable information such as names, social security numbers, or email addresses, even in hashed form."*

Flurry, another popular analytics service, has the following text in its documentation [11] for ASM `setUserID()`:

> *"**Warning**: It is a violation of our terms of service to track personally identifiable information such as a device ID (e.g. Android ID) using this method. If you have a user login that you wish to send to Flurry using this method, you must anonymize the data using a hashing function such as MD5 or SHA256 prior to calling this method."*

App developers may also attempt to reduce PII-related misconfigurations by adopting privacy policies that require anonymization or aggregation of data when used with analytic services. For example, the privacy policy for the app ShopClues[29] claims:

> *"ShopClues.com may also aggregate (gather up data across all user accounts) personally identifiable information and disclose such information in a non-personally identifiable manner to advertisers and other third parties for other marketing and promotional purposes."*

Despite such documents and policies, it is not clear whether app developers always follow them in reality as they may neglect them during development. In this paper, we perform a study to understand how app developers invoke ASMs in practice and whether those practices comply with the documents and policies of the analytic service providers and the apps themselves. It should be noted that, while there exist research efforts on data collection behavior, over-privilege, and leak detection for third-party libraries [36, 41, 43–45], our work is different in that it studies the cause of leaks related to misconfiguration of third-party services. Specifically, we try to answer the following four research questions in this study.

- **RQ1:** What configuration methods do analytic service provide and how do apps invoke those methods?
- **RQ2:** How commonly do app developers use PII when configuring analytic service?
- **RQ3:** Do analytic services provide mechanisms to protect anonymity in the case of misconfiguration as a result of **RQ2**?
- **RQ4:** Do analytic service misconfigurations result in violations of apps' own privacy policies and analytic service providers' documents/policies?

To answer these research questions we developed a semi-automatic approach, Privacy-Aware Analytics Misconfiguration Detector for Android (PAMDroid), to detect and analyze misconfigurations that may lead to privacy risk. In this approach, we first investigated the documentation of the 18 most popular analytic services in the mobile analytic ecosystem as listed in AppBrain[2]. We acquired the methods provided by these analytic services through their Application Program Interface (API) specifically for configuring user attributes (ASMs). We also collected the configuration instructions and terms of service notices from these analytic services, when available, to gather their guidelines and recommendations for use. With this data, we designed and conducted an experiment to dynamically and automatically evaluate the top 1,000 Google Play store apps that contained at least one ASM invocation in their code. We detected invocations to attribute-setting ASMs at run time and recorded parameter values to study what the common practices were and whether they abode by the app's privacy policies, the analytic service guidelines, and best practices concerning PII for using analytic services. We also investigated the analytic reports generated by the analytic services to study whether the services applied any mechanisms to anonymize or aggregate the collected data.

We have the following major findings:

- Based on the results of our semi-automated approach, 555 out of 1,000 top apps from the Google Play store had at least one ASM invocation observed at run time and 120 of them used PII to configure analytic service without encryption.
- All the analytic services we investigated provide behavior reports on individual users to app developers and the reports are labeled with exactly the same identifiers provided by app developers. Therefore, if PII is used as an identifier, they will be directly linked to the user behavior reports, resulting in targeted, non-anonymous and non-aggregated information.
- We manually inspected the policies of the 120 apps and found 27 of them may violate their own privacy policies by using PII as user attributes.
- Using PII with analytic services may also violate the Terms of Service (TOS) of analytic services. Among the analytic services we studied, we found that four of them explicitly require app developers to avoid passing PII to ASMs. They are Firebase, Google Analytics, Flurrry and Mixpanel, and they have the app-market shares of 55.95% [23], 26.84% [25], 5.12% [24], %0.77 [26], respectively. Although only four analytic services state this requirement explicitly, Firebase, Google Analytics, and Flurry are the top three market share

Table 1: Analytic services collect user events by default

Firebase	ad_click, ad_exposure, ad_impression, screen_view, user_engagement, session_start, app_clear_data, app_exception, etc.[7]
Mixpanel	first app open,app updated, app crashed, app session in app purchase. [18]

holders and dominate the market, so we believe this requirement is a standard for analytic services. Our result shows that 37 apps which are using the four analytic services did set user's PII to the ASMs, and thus may violate analytic services' terms of service (Firebase, Google Analytics, Flurrry) or privacy guidelines (Mixpanel).

2 BACKGROUND ON ANALYTIC SERVICES

For a better understanding of users' behavior, app developers often choose to utilize analytic services. Analytic services usually provide client libraries that app developers could utilize in their app, which will record an app user's interaction with the app and send the corresponding data to the server of the analytic service. Later, the analytic services can link the activity of a mobile app user over time into a behavior report. The behavior report includes detailed usage information about this user. The analytic services can then aggregate all the users' reports and provide analytic data to the app developers so that they can improve their product or make better business decisions based on the analytic report.

In this section, we describe the background information about analytic services, especially about the user events they track, their attribute setting methods and terms of service.

2.1 Tracked User events

Analytic services automatically collect some events that are triggered by basic interactions such as ad impressions, ad clicks, and screen transitions. Table 1 shows the default events collected by Firebase and Mixpanel. From the table, we can see that the collected events contain detailed information about the user's usage of the app and interactions with the ads.

2.2 Analytic Service Configuration

Analytic services provide *Attributes Setting Methods (ASMs)* that enable developers to customize the analytic service by setting some attributes for their users. Developers can set identifiers or other attributes such as age, gender, and location on each app user. Later, developers can use those attributes as a filter or metrics in their analytics reports. For instance, a developer may want to know the geography distribution, or age distribution of their users. The data that developers pass to those ASMs will be associated with the users' collected events and then sent to the server of analytic services. To protect users' privacy, analytic services have certain guidelines or suggestions for how the developer should use those ASMs. We list two from some analytic services here as examples:

In Firebase[10] [8]:

> When you set user properties, be sure to never include personally identifiable information such as names, social security numbers, or email addresses, even in hashed form.
> Note: You are responsible for ensuring that your use of the user ID is in accordance with the Google Analytics for Firebase Terms of Service. ... For example, you cannot use a user's email address or social security number as a user ID.

In Mixpanel[19]:

> If you wish to track users truly anonymously, however, then your tracking implementation should not use user-specific information, such as the user's email address. Instead use a value that is not directly tied to a user's PI (personal information), whether it be a unique anonymous hash, or a non-PI internal user identifier.

These instructions require the app developers to not use any PII to configure analytic services and encourage them to use anonymous data instead.

2.3 Personally Identifiable Information

We consider PII as the union of the definitions by Google Analytics and the EU General Data Protection Regulation (GDPR). The following statement is from Google Analytics [31].

> "*Google interprets PII as information that could be used on its own to directly identify, contact, or precisely locate an individual. This* ***includes****: email addresses, mailing addresses, phone numbers, precise locations (such as GPS coordinates - but see the note below), full names or usernames*"

The following statement is from GDPR [14].

> "***Personal Data:*** *... an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.*"

GDPR also defines online identifiers [16] which we include as PII:

> "***Online Identifiers:*** *Natural persons may be associated with online identifiers provided by users' devices, application, tools or other identification tag and it could be used to associate with natural persons, because online identifiers may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them.*"

3 PAMDROID AND STUDY DESIGN

The goal of this research is to detect misconfigurations in analytics services as they may lead to privacy risks. To this end, we developed

PAMDroid, a semi-automated approach to detect the misconfiguration of analytic services due to setting PII to ASMs. As illustrated in Figure 1, there are two manual preparation steps of PAMDroid. First, we manually collect a set of most popular analytic services and Android apps. For each analytic service, we investigate its API documentation to collect all ASMs that app developers can use to set user attributes. Second, we set up an Android device and collected all its information to construct a *reference user profile*. The profile includes different platform IDs (e.g., device ID, serial number, Android ID, advertising ID), a synchronized Google account (e.g., user name, user email, address, age, gender, date of birth), and other sensitive information (e.g., location, IP address, MAC address).

After these two steps, PAMDroid first performs static smali code analysis on the apps to filter out the apps that do not invoke any ASMs at all. Then, PAMDroid automatically instruments all ASMs (detected with static smali code analysis) to print their argument values to system log. After that, PAMDroid uses Monkey[30] to test the instrumented apps' user interface. Note that many apps trigger analytic services only after a user is logged in. As a supplement of Monkey, we perform manual login for all apps that require login to get to the start page. Finally, PAMDroid compares the collected system logs with the reference user profile. When certain types of information in the reference user profile show up in the system log, PAMDroid detects an ASM misconfiguration. After all misconfigurations are detected, we manually inspect the corresponding apps' privacy policies and corresponding analytic services' terms of services to detect violations and misalignments. It should be noted that the major goal of this research is to study the commonality and characteristics of ASM misconfigurations, and PAMDroid is developed for the study, so we supplemented it with manual analysis to acquire most comprehensive and accurate results. If we do not perform manual log-in and adopt existing automatic approaches for policy analyses [46, 49, 50, 52], PAMDroid can be made fully automatic, but its effectiveness is not clear and it is not the focus of this paper.

We next introduce the details of our study setup with the PAMDroid approach.

3.1 Collection of Apps and Analytic Services

We identified the 18 most popular analytic services using published statistics provided by AppBrain [2], a company specializing in app marketing and promotion. After that, we identified the ASMs provided by the selected analytic services. The top 1,000 free apps containing at least one invocation of the studied ASMs were collected from PlayDrone [20], a collection of metadata for Android apps on the Google Play store. We identified those apps which invoked ASMs by analyzing their smali code[2]. If an app obfuscated the ASMs it invoked, we could not apply our approach to it. Furthermore, we also ruled out apps that were incompatible with our device and those no longer existing in Google Play due to being removed since being included in the PlayDrone database.

To determine whether an app had an invocation of a studied ASM, we first decoded the analytic libraries into smali format using APKTool [47] and identified each ASM's smali signature. We then decompiled each app's APK (Android Package) file into smali format and scanned the resulting file for occurrences of ASM signatures. Only apps containing at least one ASM signature were kept for consideration.

[2]Assembler for the dex format used by Dalvik

3.2 Runtime Information Collection

There are multiple approaches to detect information flow to ASMs. The first approach we considered was using static taint analysis. To this end, we used FlowDroid [32] to analyze the 1,000 apps and defined ASMs as sinks and personal information sources from SuSi [39] as sources. The result showed that FlowDroid only identified 10 data flows from sources to sinks. Through further investigation, we found that the sources of PII sent to ASMs are often not Android API methods, but system files or databases. Furthermore, PII often flow through paths that are not handled by FlowDroid, such as `android.content.SharedPreferences`, which is a data structure in Android system that stores user information such as username, device ID, etc. If we add all these API methods as sources of FlowDroid, it will report many false positives as files, databases, and Android system data structures may also contain a lot of non-PII.

To make sure our study is conservative (all reported misconfigurations are real), we ultimately utilized value-based dynamic taint analysis. As mentioned earlier, we prepared a reference user profile to match arguments sent to ASMs. To make sure values in our user profile are not confused with other values, we designed very strange information (e.g. user name, email address) for the synchronized Google account. To make sure our matching is robust, for the values in the reference user profile, we further generate values with different value transformations, such as reverting and truncating. We also produced hashes for all PII using common hashing algorithms provided by Android API methods so that we could identify hashed values (although in the study we did not find hashes being sent). Note that we manually confirmed all matched results to make sure that our value-transformations do not lead to wrong matches. One limitation of value-based taint analysis is that we cannot detect encrypted PII with an app-specific key. Notably, using encrypted PII as user attributes on analytic service already reduces the risk to privacy, because the unencrypted PII will not be combined with collected user behavior.

In order to catch the arguments of ASM invocations during runtime, we instrumented all ASMs in smali code by adding a call to the Android logger to report the invocation at the beginning of the ASM implementation. This allows us to use the Android system log to analyze method argument values being set at runtime. After inserting the code, we rebuilt the smali code back into APK format for testing. We used the Android Debug Bridge (adb) to automatically install the rebuilt apps onto our test device and run the apps and then executed Monkey to perform the testing. For each app, we automatically installed, executed, tested, uninstalled and saved the system log into the local file system for later inspection. During testing, we found 254 apps requiring login to an account to show the app's start page, so we manually created accounts for these apps using the reference user profile to complete the login process.

Finally, PAMDroid searched the system logs generated during testing and extracts argument values of ASMs based on flags inserted during instrumentation. Table 2 is an example where Line

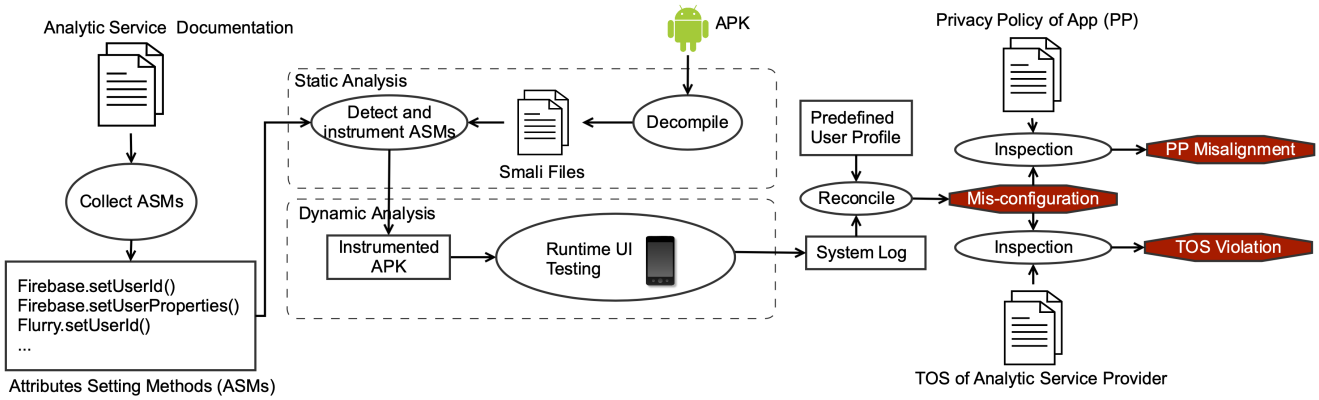

Figure 1: Privacy-Aware Analytics Misconfiguration Detector(PAMDroid)

1 shows our inserted flag; Line 2 shows the ASM that be invoked (`Firebase.setUserProperty`), and Line 17 shows our flag and the first argument value that was passed to the ASM ("vivino_email"). Line 18 shows the second argument value which was the email address (represented as "**********@gmail.com").

4 STUDY RESULTS

In this section, we present the results of our study and answer the research questions.

4.1 Apps' Usage of Analytic Services

To answer **RQ1**, for each analytic service, we first investigated their documentation and collected the ASMs. We noticed that every analytic service provides the methods that allow developers to set attributes for their users, such as `setUserID`, `setCustomerUserId`, or `setUserIdentifier`, etc. Firebase provides an method called `setUserProperty`, which allows developer to set any attributes to describe their user. It takes two arguments which are similar as a pair of "key" and "value". Other methods include `setUserEmail`, `setLocation`, `setAge`, `setGender`, `setDeviceId`, `setPhoneNumber`, etc. The full list of ASMs are available at our anonymous project website [3]. Four analytic services (Firebase, Google Analytics, Flurry, Mixpanel) explicitly require app developers to avoid setting PII [8, 9, 11, 19] to ASMs.

A method to set user identifier (e.g., `setUserId`) is provided by every analytic service and mostly commonly invoked in our test. For example, `Crashlytics.setUserIdentifier` was invoked in 147 apps, and `Flurry.setUserId` was invoked in 67 apps. We present these frequencies in Table 3. In the table, the first column presents the analytic service name; the second column presents the total number of apps that invoked the ASMs from this analytic service. The third column represents the ASM name; and the fourth column presents the number of apps that invoked the corresponding ASM. Among the 1000 apps that contain ASM invocations in their smali code, 555 apps invoked 29 ASMs from 13 different analytic services during our runtime testing. Table 3 shows that Firebase and Crashlytics are the most commonly invoked analytic services. Note that a single app could use more than one analytic services, within one analytic service, the app could invoke multiple ASMs to set user attributes.

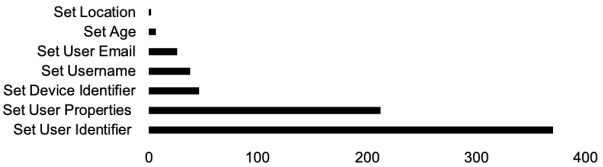

Figure 2: #Apps Invoking Different Types of ASMs

To understand how apps use different types of ASMs over all analytic services, we categorized all ASMs in Table 3 into a number of categories according to their purposes. In particular, the categories are "set user identifier", "set user properties", "set device identifier", "set user email", "set username", "set age", and "set location". In Figure 2, we present the number of apps that invoke different categories of ASMs. We observed that 387 apps set user identifiers to at least one analytic service, showing that many app developers set identifiers for users to differentiate individual user interactions through the analytic service, and the function is also well supported by analytic services in general. Furthermore, 198 apps set user properties to at least one analytic service. Since ASMs in the "set user properties" category are very general and can be used to set almost any data, it is difficult to statically tell what information is sent through them.

Finding 1. Our answer to **RQ1** is that, all our studied analytic services provide ASMs for app developers to set user attributes, and more than half (555 of 1,000) of apps trigger ASMs to label user behavior reports.

4.2 PII set to ASMs in Misconfiguration

To answer **RQ2**, we further studied what types of data are set to ASMs in our studied apps. By matching the logged method

[3] https://sites.google.com/site/trackersec2019/

Table 2: System log of ASM invocation

```
1  01-10 18:16:55.024 1931 1931 W System.err: java.lang.Exception: Third-party API invoke detection:Print StackTrace with parameter:
2  01-10 18:16:55.024 1931 1931 W System.err: at com.google.firebase.analytics.FirebaseAnalytics.setUserProperty(Unknown Source)
3  01-10 18:16:55.024 1931 1931 W System.err: at com.vivino.android.a.a.a(FirebaseHelper.java:160)
...
...
17 01-10 18:16:55.024 1931 1931 I Third-party API invoke detection:Print StackTrace with parameter:: vivino_email
18 01-10 18:16:55.024 1931 1931 I Third-party API invoke detection:Print StackTrace with parameter:: **********@gmail.com
```

Table 3: #Apps Invoke Different ASMs

Analytic library	# Apps	Method	# Apps
Firebase	216	setUserId	64
		setUserProperty	193
Crashlytics	163	setUserEmail	21
		setUserIdentifier	147
		setUserName	38
AppsFlyer	81	setAndroidIdData	31
		setAppUserId	4
		setCustomerUserId	63
Flurry	70	setAge	6
		setLocation	2
		setUserId	67
Tune	38	setAndroidId	12
		setDeviceId	3
		setUserEmail	2
		setUserName	2
		setUserId	27
		setFacebookUserId	6
		setGoogleUserId	6
		setTwitterUserId	6
IronSource	24	setUserID	24
mixpanel	17	identify	17
Applovin	13	setUserIdentifier	13
Leanplum	12	setDeviceId	6
		setUserId	4
		setUserAttributes	5
Branch	11	setIdentity	11
Google Analytics	7	setClientID	7
Appsee	6	setUserId	6
Newrelic	4	setUserId	4

arguments to the controlled user profiles (see Section 3.2), we can detect misconfigurations on the fly.

Table 4 presents the number of apps setting different types of PII to ASMs. In particular, Columns 1-4 present the type of PII, ASM name, the number of Apps setting certain type of PII to a certain ASM, and the total number of Apps setting certain type of PII to all ASMs. We make three major observations. First, overall 120 apps set PII or PII's transformation (11 apps) to ASMs. It should be noted that a single app may set multiple data types, so the values in Column 4 do not add up to 120. Second, among the 120 apps, 79 apps set Android ID to ASMs, 24 apps set users' email addresses to ASMs, and 19 apps set users' registered username to ASMs. Note that registered usernames are used to uniquely identify users in the app, and many users use the same username across apps, so Google Analytics explicitly lists username as PII [22]. Third, one type of PII is observed to be set to ASMs for multiple purposes. For example, Android IDs are mainly set to ASMs in the category "set user identifier", but it is also set to `Crashlytics.setUserName()` and `Firebase.serUserProperty()`. Email addresses are also set to ASMs in the categories of "set user properties" and "set user identifier". So the vagueness and generality of ASM design may have aggravated their misuse.

In Figure 3, we further show the number of apps that set different PII to different analytic services. In the figure, we organize the number of apps setting various PII to each analytic service as a separate column chart. In each sub-column-chart, the x-axis shows different analytic services, and the y-axis shows the number of apps setting different personal information type in that analytic service. From the figure, we can see that Crashlytics and AppFlyer are receiving PII from the most number of apps, and Crashlytics also received user email addresses from the most number of apps. Furthermore, Firebase and Flurry, which explicitly require app developers to not send PII to them, both receive various types of PII, including Android ID, device series number, and username. Firebase further receives email address, and Flurry further receives IMEI.

Finally, Figure 4 presents the category distribution of our dataset and the percentage of apps (in each category) setting PII to ASMs. Each bar represents the total number of apps in the specific category, while the dark portions represent the number of apps in the category that set PII to the ASMs. We further label the percentage of dark bar portion for each bar. The figure shows that there is not a specific category of apps that are much more likely to use PII as user attributes. Compared with others, apps in Photography, Communication and Shopping have higher possibility of setting PII to ASMs. Besides PII, our test result shows that 24 apps used Advertising IDs, which can be changed by users and sometimes encouraged by analytic services to be used as user identifiers. However, if users do not change Advertising IDs frequently, they can still be actually PII. Since we want our study results to be conservative, we do not include them as PII in our study results.

Finding 2. Our answer to **RQ2** is that, among the 1,000 apps we studied, at least 120 apps (detected by PAMDroid) misconfigure ASMs with PII. In particular, Android ID (in 79 apps), User Email (in 24 apps), Username (in 19 apps), IMEI (in 6 apps), and Serial Number (in 3 apps) are the types of PII being set to ASMs.

4.3 Enforcement of Aggregated and Anonymous Reports

To answer **RQ3**, we studied all 13 analytic services being invoked to find out whether they have enforcement mechanisms to reject PII being set to ASMs. Unfortunately, none of 13 services have such built-in enforcement mechanisms. Only one of them, Appsflyer [4],

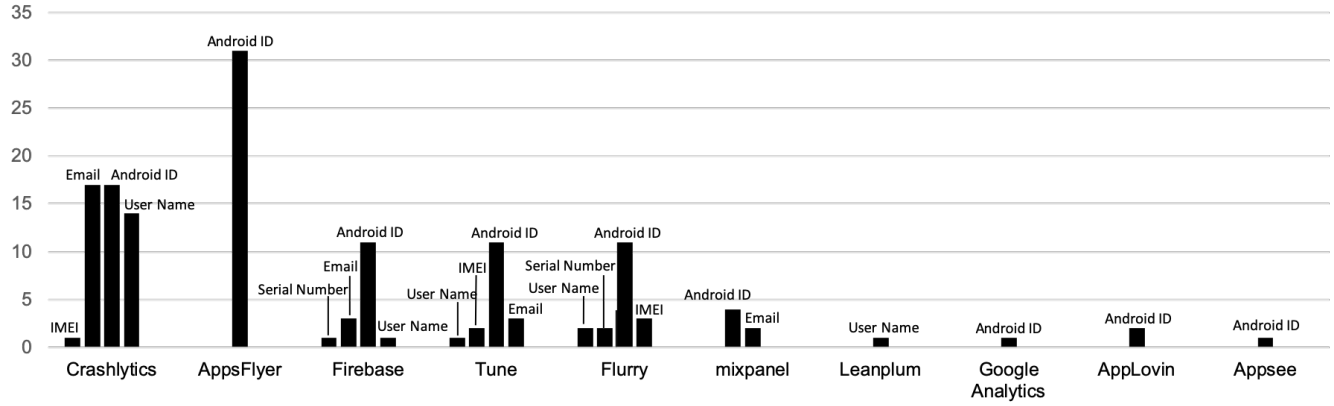

Figure 3: # Apps set different PII to different analytic services

Table 4: #Apps Setting Different PII to ASMs

Personal Info	Tracker API	#Apps	Total
Android ID	Firebase.setUserId	8	79
	Firebase.serUserProperty	5	
	AppsFlyer.setAndroidIdData	29	
	AppsFlyer.setCustomerUserId	2	
	Flurry.setUserId	11	
	Mixpanel.identify	4	
	Tune.setAndroidId	11	
	Tune.setDeviceId	1	
	Crashlytics.setUserIdentifier	16	
	Crashlytics.setUserEmail	1	
	Crashlytics.setUserName	1	
	Applovin.setUserIdentifier	2	
	GoogleAnalytic.setClientId	1	
	Appsee.setUserId	1	
Email	Firebase.serUserProperty	3	24
	Mixpanel.identify	2	
	Tune.setUserEmail	2	
	Tune.setUserName	1	
	Crashlytics.setUserEmail	12	
	Crashlytics.setUserIdentifier	1	
	Crashlytics.setUserName	5	
Username	Firebase.serUserProperty	1	19
	Flurry.setUserId	2	
	Tune.setUserName	1	
	Crashlytics.setUserName	14	
	Crashlytics.setUserIdentifier	1	
	Leanplum.setUserAttributes	1	
	Leanplum.setUserId	1	
IMEI	Flurry.setUserId	3	6
	Tune.setDeviceId	2	
	Crashlytics.setUserIdentifier	1	
Serial Number	Flurry.setUserId	2	3
	Firebase.serUserProperty	1	

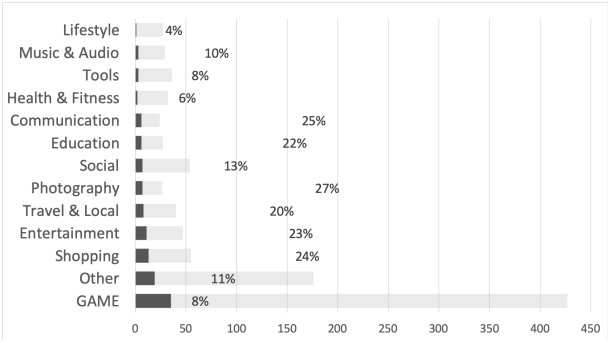

Figure 4: # Apps distribution in categories

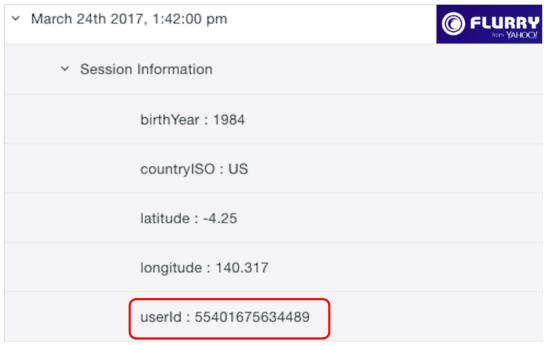

Figure 5: A demo report in Dashboard of Flurry [12]

provides a method to set user email address with encryption, but none of apps in our data set actually invoked this method. Furthermore, we studied whether the information set to ASMs is encrypted before they are combined with behavior reports, and no analytic service is performing the encryption. It should be noted that all the analytic services which we studied use encrypted network connection (e.g., HTTPS) to send collected information. However, if the PII set to the ASMs is combined with behavior reports in un-encrypted form, the anonymity of the collected user behavior is already lost as the whole data will be decrypted later.

It is very challenging to tell how data is stored and processed on servers of analytic services. However, we can predict their practice from the behavior reports they provided to developers. Therefore, we further studied whether the analytic services provide reports on individual user behaviors. We found that for all analytic services that we investigated, their online analysis reports for developers are

not limited to aggregated data, but are instead itemized by received user attributes. Figure 5 , Figure 6 and Figure 7 presents example report screen-shots from Flurry, Mixpanel, and Crashlytics. From the three figures, we see that reports are organized by user attributes and presented to app developers, and the identifiers (e.g., user email, username or device IDs) are presented without anonymization. Figure 5 shows that Flurry's report not just contains the userId, but also user's latitude and longitude data.

Finding 3. Our answer to **RQ3** is that, analytic services do not have any mechanisms to vet or anonymize PII they received from ASMs. The PII are directly combined with behavior reports when stored and provided to app developers.

4.4 Policy Violations and Misalignment

We present our answer to **RQ4** in this subsection. As we discussed in our results above, it is a privacy risk when PII was set by app developers on analytic services without encryption or anonymization. Such misconfiguration may cause two types of policy-related issues. First, to protect user privacy and avoid legal liabilities, analytic services may state in their TOS that they do not allow developers to set PII to their ASMs. So the misconfiguration of ASMs will cause TOS violations. Second, the app's own privacy policy may claim anonymous data analytics or fail to describe the sharing of PII to analytic services, so the misconfiguration of ASMs will cause misalignment between code and privacy policies.

4.4.1 TOS Violations. Figure 3 shows that 120 apps set PII on ten analytic services. As we mentioned in Section 1, four analytic services (Firebase, Google Analytics, Flurry, Mixpanel) explicitly require app developers to avoid setting PII to ASMs in their terms of service (Firebase, Google Analytics, Flurry, note that they are the top three market-share holders in analytic services) or privacy guidance (Mixpanel).

Based on our experiment results, 31 apps have set PII to ASMs of Firebase, Google Analytic, or Flurry, and thus we believe that the misconfiguration of ASMs actually violates their terms of services. Furthermore, 6 apps have set PII to ASMs of Mixpanel, so they are violating Mixpanel's privacy guidance. It should be noted that, although the remaining 83 apps did not violate the policies of analytic services, their practice of setting PII to ASMs still jeopardizes users' privacy. Also, the top 3 market share holders have relatively less (31/120) misconfigurations maybe because they have instructions of ASMs in their documentation and TOS, which help avoid misconfigurations.

4.4.2 Misalignment of Apps' Privacy Policies. Misconfiguration of ASMs may also cause misalignment between an app's code and privacy policy. To find such apps, for each of the 120 apps that set PII to ASMs , three of the authors independently read the app's privacy policy and wrote arguments on why he / she believes using PII for analytic services is a potential policy misalignment or not. Then, the authors met to discuss the arguments for each app, and voted to determine whether the misconfiguration is misaligned with the privacy policy.

We found 27 out of 120 apps have misconfigurations that are misaligned with their own privacy polices. 15 apps vaguely mentioned in their privacy policy that they may share PII of users with third parties. 58 apps have no misalignment with their own privacy policies as they explicitly indicate that they will share specific personal information type to third-parties. The remaining 20 apps either have a non-English privacy policy or the privacy policy web-page is not available. The detailed discussion record of all 120 apps is available in our anonymous website, and misalignment examples are presented later in this subsection.

Privacy Misalignment. We consider an app to be misaligned with its privacy policy if the policy does not indicate that it will share PII with third parties, or if the policy claims anonymous data collection. For example, the social app Emojidom's privacy policy [27] states that:

> ***Do third parties see and/or have access to information obtained by the Application?***
> *Only aggregated, anonymized data is periodically transmitted to the analytics tools which help us evaluate the Application's usage patterns and improve the Application over time.*

However, our test results show that this app set user email addresses to Crashlytics, which is misaligned with this privacy policy.

Vague Privacy Policies. Privacy policies should inform users about types of user information are shared with third parties. Third party analytic services also request app developers to make this sharing explicit in their apps' privacy policies. For example, Crashlytics is one of the most popular third party analytic services for helping developers to analyze crashes in their apps. Crashlytics requires that all developers maintain a privacy policy that fully and accurately discloses the type of information shared with Crashlytics [21]. Among 120 apps that send PII to analytic services, 15 of them abstractly indicate that they may share personal information to third-parties without specifying what the information types are. For example, the shopping app Staples sets user email address to Crashlytics and its privacy policy states that [28]:

> *We may share your Personal Information with our third-party service provider to process transactions or provide services on our behalf, including but not limited to providers of product delivery services (for example UPS and FedEx) and website analytics (for example Google Analytics).*

No misalignment. We consider an app has no misalignment with its privacy policy if it clarifies the data types being shared with third-party service providers.

Finding 4. Our answer to **RQ4** is that, among 120 apps with misconfiguration of ASMs, the misconfigurations cause terms-of-service violation of analytic services in 31 apps, and privacy policy misalignment in 27 apps.

4.5 Threats to Validity

The major threat to internal validity of our study is the false positives and negatives in our misconfiguration detection process. Since

Figure 6: A demo report in Dashboard of Mixpanel [33]

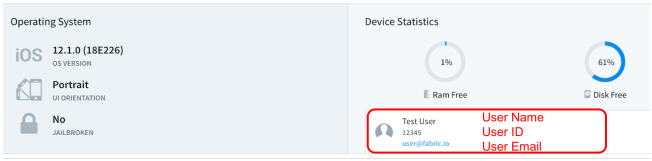

Figure 7: A demo report in Dashboard of Crashlytics [5]

we report only observed misconfigurations at run time, we should not have false positives. It is possible that our dynamic analysis failed to trigger some misconfigurations, our collected ASMs are not complete, or our matching process missed some sophisticated transformed argument values. So our reported number of misconfigurations is actually an under-estimation, which will not undermine our major findings. To reduce this threat, we carefully scanned the documentation of analytic services, combined monkey and manual log-in to enhance the code coverage, and considered various value transformations when matching the reference user profile with system logs. Since most developers will perform configuration of analytic libraries when the app is started, the false negative rate caused by uncovered misconfiguration should not be high. The major threat to external validity of our study is that our findings may apply to only the 1,000 apps under study. To reduce this threat, we chose the top apps from Google Play store and these apps covers almost all different app categories.

5 LESSONS LEARNED

In this section, we discuss the potential privacy risks found and our recommendations for different parties involved in the configuration of analytic services.

5.1 Privacy Risks

Although top analytic services advise app developers to not use PII as user attributes, many app developers still do so and no mechanism has been provided (either by Android or the analytic services themselves) to prevent app developers from using PII. This means that the analytic services may unintentionally link a behavior report to a specific individual. Based on our experiment results, a non-trivial number of apps are using emails and device identifiers (e.g., Android ID, IMEI, serials number) as user attributes. These identifiers are long-lived and can be used to construct a user's comprehensive profile from multiple apps using the same analytic service. Since most analytic services further share their collected data to third parties for business purposes, the personal-identifiable comprehensive profiles can be exposed to more risk due to the neglect of PII inside the data.

Since analytic services and app developers hold a large amount of valuable user data, it is very likely that they can be targeted for information theft/leakage attacks. When an information leakage incident happens, if the data stored on the server is not in an anonymous and aggregated form, the consequence will be much more severe than the scenarios where they are anonymized and aggregated. Because analytic services do not expect app developers to set PII to the ASMs, they may not have corresponding mechanism to detect PII in the collected data, and thus may not use protection mechanisms (e.g., encryption) on the collected data.

5.2 Actionable Suggestions

Base on our study, the five parties involved in analytic services may take some counteractions to reduce the privacy risk caused by ASM misconfigurations.

Research Community In order to precisely and comprehensively detect misuse of analytic service ASMs, new static techniques are desired to detect the data flow from PII sources to the ASMs. Although it is possible to take advantage of off-the-shelf information flow analyses [32, 34], the challenge still remains of detecting PII sources and ASMs. For PII sources, many types of PII (e.g., username, user's email) are user defined so their source may be a text box, a local file or a database which cannot be easily differentiated from other non-PII information. Therefore, more precise techniques to identify PII sources or intermediate sources (e.g, a variable that loads PII values from a file or the database) are required. For ASMs, although we manually constructed ASM sets for 18 popular analytic services in the study, the analytic services are continuously evolving and new analytic services may become popular. For this reason, our sets can quickly become out-of-date. Therefore, novel techniques to automatically identify ASMs and their behaviors are desirable.

Another potential research endeavor is studying how analytic services can vet and anonymize PII so that they can enforce the privacy requirement of using ASMs. One challenge is that the analytic services do not know where the argument values come from. So, a likely solution is value-based detecting of PII, where a classification model may be learned to detect PII values in run-time arguments.

Privacy profiles [41] are automatically extracted from apps to provide fine-grained information of collected and shared information types, but they cannot handle advanced privacy properties such as data anonymity and aggregation. Anonymity may be verified by checking whether data is combined (e.g., concatenated, put into one object or key-value pair) with PII. Aggregation may

be verified by checking whether individual data is destroyed (e.g., freed) at the end after they are read.

App Developers. App developers should take more care in following ASM documentation/terms of service and avoid setting any PII as user attributes. Instead of using raw PII, developers could encrypt or hash the data before it is passed to analytic services or use non-PII instead. For example, if the differentiation of users helps on more precise statistics (e.g., how many users are using their app or certain activity), they can use Advertising IDs, randomly-generated IDs, or encrypted/hashed PII as user's identifier. App developers should pay attention to their privacy policies as well, as they need to make sure the policy is consistent with their practice of using analytic services. At the same time, a clear profile on what kind of PII is set to ASMs can help users understand how their privacy data can be used by analytic services.

Analytic Services. Analytic service developers should enhance and enforce data anonymity and aggregation in their code base. In particular, just like Google Analytics, Firebase, Flurry, and MixPanel, other analytic services should also try to provide a more clear and easily reachable instruction about privacy-aware configuration. Meanwhile, when designing and implementing methods, analytic service developers should avoid or limit the usage of overbroad/vague methods (e.g., `setProperties()`) and methods that are meant to receive PII (e.g., `setUserEmail()`, `setUserLocation()`). They should also add encryption features for methods that may receive PII from the app.

Second, when an app sets PII to ASMs, analytic services could have mechanisms to detect and anonymize the PII (e.g., regular expressions). In this way, analytic services could add vetting mechanisms in the implementation of ASMs to reject PII or raise warnings on detection. Alternatively, instead of transparently handing over the PII to app developers in their reports, they could encrypt the PII or replace it with other non-PII, and then perform analysis on the pre-processed data. After that, analytic services should generate a report that only contains aggregated data about user behaviors.

Platform Providers. The Android platform has applied some strategies to reduce the privacy risk over the years. For example, in Android version 8.0 and higher, Android ID is no longer a constant value for different apps installed on the same device. This mechanism helps to prevent the analytic services from gathering an individual user information across multiple apps. Since the Android platform has access to much PII for the device's users, such as Google email account, Android ID, Device ID, it should be able to vet such data sent to ASMs of analytic services. Working together with analytic services (e.g., asking them to annotate ASMs), the Android SDK could provide on-the-fly suggestions on which APIs and API options should be used while app developers are coding. Furthermore, the Android platform could provide the option to automatically reset the Advertising ID periodically for users.

App Users. App users should be aware that they can be un-anonymously tracked if app developers do not properly set their attributes on analytic service. Our study found that some app developers use usernames in analytic services so we suggest app users to avoid using their real names or PII when registering with different apps. In addition, Google Advertising ID has been encouraged to be used as the individual identifier in the analytic services. However, if a user does not reset the Advertising ID frequently, it becomes another long-lived online identifier. So we encourage app users to reset their Advertising IDs periodically to avoid being identified as the same individual for a long time period.

6 RELATED WORK

In this section, we categorize related existing research efforts into the following three categories: studies on the data collection and sharing of analytic services, general information-leak detection techniques, and privacy policy analysis.

6.1 Data Collection and Sharing of Analytic services

Existing research efforts mainly studied what user activities are tracked by analytic services and what information they may collect. Liu et al.[37] investigated the types of user activities being tracked by analytic services. Their results reveal different levels of user-activity tracking on different UI event types. Since analytic libraries are integrated into the app, they receive the same privilege (e.g., permissions) of the enclosing app from the Android platform. This allows the analytic services to collect some personally sensitive device information. Seneviratne et al. [44] show that 60% of paid apps are connected to analytic services that collect personal data compared to 85% - 95% of free apps. They perform static analysis on Android API calls inside the analytic libraries and summarize the type of personal data collected by the analytic services from the Android platform. Compared with these works, our approach focuses on misconfiguration of ASMs where PII can be combined with user behavior reports to compromise their anonymity. This is a novel aspect that has never been investigated in the above efforts.

6.2 Detection of General Information Leaks

There has been a lot of work on the detection of information leask on mobile platform. In particular, ClueFinder [38] leverages NLP technology for building a learning system to identify sensitive data leaks from apps to third parties. FlowDroid [32] leverages static taint analysis with tunable sensitivity to trace information from sources to sinks so it can also be used to detect information leaks. TaintDroid [34] is one of the most popular Android taint systems for tracking the information flow. Their study shows that two-thirds of apps introduce potential privacy risks to sensitive user data. VetDroid [51] is a dynamic analysis platform to construct permission use behavior during runtime by intercepting the invocations of Android APIs, which can be used to analyze information leaks. Han et al. [35] uses dynamic taint analysis to study how apps expose personal data and persistent identifiers in information flow. They present a prototype privacy control, which inserts code checks at all Android API invocations that access sensitive data. Network traffic analysis techniques have also been applied to detect personal data that app share with third parties [41] [43][42]. Razaghpanah et al. [40] detect third-party advertising and analytic services at the traffic level. Ren et al. [43] instrument VPN servers to identify privacy leaks in network traffic. Vallina et al. [48] analyze mobile ISP traffic logs to identify advertisement traffic. Compared with these works,

our PAMDroid also uses value-based dynamic taint analysis to detect information leaks. However, our major contributions include identifying the ASM misconfiguration problem, and the construction of ASM sets for popular analytic services. We also performed a study to reveal the severity of the ASM misconfiguration problem in practice.

6.3 Privacy Policy Analysis

Privacy policies inform users on how their information will be collected, used and disclosed. Existing works have been working on detecting misalignment between privacy policy and the actual data practice in app code [46, 49, 50, 52]. They analyzed the app code and detected what sensitive information types from user input or Android platform API invocations are sent to network. After that, they compared the collected and shared with information types with the statements in privacy policies. Different from these previous studies, our work tries to investigate whether developers' practice on analytic services configuration may compromise anonymity and aggregation of users' behavioral reports. We developed PAMDroid to detect miconfigurations of ASMs, and our study shows that a non-trivial number of apps set PII to ASMs of analytic services. As a result, our work further detects TOS violations of analytic services and privacy-policy misalignment related to anonymity and aggregation, which are never reported by above research efforts.

7 CONCLUSIONS

In this paper, we developed a semi-automated approach PAMDroid to investigate whether mobile app analytic services are really anonymous as they are often claimed and how ASMs can be misconfigured by app developers. Our study on 1,000 popular apps has shown that most analytic services provide ASMs, such as `setUserId()`, to differentiate users. These ASMs can be misconfigured by developers so that individual user behavior profiles can be disclosed, which might impose greater privacy risk to users. We found that misconfiguration of ASMs in 37 apps leads to violations of analytic services' terms of service, and misconfiguration of ASMs in 27 apps leads to privacy policy misalignment. In future, we plan to further study what user behaviors are collected by analytic services besides the events collected by default and to investigate whether PIIs can also be leaked through user events. Moreover, We are going to develop a fully automated framework to detect PIIs being set to ASMs without encryption.

ACKNOWLEDGMENTS

This work is supported in part by NSF Awards 1748109, 184646, 1453139, 1948244 and 1736209.

REFERENCES

[1] 2017. *Equifax Data Breach.* Retrieved May, 2019 from http://fortune.com/2018/09/07/equifax-data-breach-one-year-anniversary/
[2] 2018. *AppBrain Android analytics libraries.* Retrieved October, 2018 from https://www.appbrain.com/stats/libraries/tag/analytics/android-analytics-libraries
[3] 2018. *AppBrain, Firebase.* Retrieved October, 2018 from https://www.appbrain.com/stats/libraries/details/firebase/firebase
[4] 2018. *AppsFlyer provide encryption option in API setUserEmails.* Retrieved October, 2018 from https://support.appsflyer.com/hc/en-us/articles/207032126-AppsFlyer-SDK-Integration-Android
[5] 2018. *Crashlytics dashboard.* Retrieved October, 2018 from https://stackoverflow.com/questions/34888420/crashlytics-how-to-see-user-name-email-id-in-crash-details/
[6] 2018. *Facebook Data Breach.* Retrieved May, 2019 from https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html
[7] 2018. *Firebase collect user event by default.* Retrieved October, 2018 from https://support.google.com/firebase/answer/6317485?hl=en&ref_topic=6317484
[8] 2018. *Firebase set user ID.* Retrieved October, 2018 from https://firebase.google.com/docs/analytics/userid
[9] 2018. *Firebase set user preoperties.* Retrieved October, 2018 from https://firebase.google.com/docs/analytics/android/properties
[10] 2018. *Firebase user propertise.* Retrieved October, 2018 from https://support.google.com/firebase/answer/6317519?hl=en&ref_topic=6317489
[11] 2018. *Flurry API setUserId().* Retrieved October, 2018 from https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/technicalquickstart/android/
[12] 2018. *Flurry dashboard.* Retrieved October, 2018 from https://developer.yahoo.com/flurry/docs/analytics/lexicon/eventreporting/
[13] 2018. *GDPR Anonymous Data.* Retrieved January, 2020 from https://gdpr-info.eu/recitals/no-26/
[14] 2018. *GDPR definition of personal data.* Retrieved October, 2018 from https://gdpr-info.eu/art-4-gdpr/
[15] 2018. *GDPR Lawfulness of processing.* Retrieved January, 2020 from https://gdpr-info.eu/art-6-gdpr/
[16] 2018. *GDPR online identifiers for profiling and identification.* Retrieved October, 2018 from https://gdpr-info.eu/recitals/no-30/
[17] 2018. *Marriott Data Breach.* Retrieved May, 2019 from https://www.consumer.ftc.gov/blog/2018/12/marriott-data-breach
[18] 2018. *Mixpanel collect user event by default.*
[19] 2018. *Mixpanel's rule about using API.* Retrieved October, 2018 from https://help.mixpanel.com/hc/en-us/articles/360000679006-Managing-Personal-Information
[20] 2018. *PlayDron metadata.* Retrieved August, 2018 from https://archive.org/details/android_apps&tab=about
[21] 2018. *Privacy policy of Crashlytics.* Retrieved October, 2018 from https://try.crashlytics.com/terms/privacy-policy.pdf
[22] 2018. Universal Analytics usage guidelines. https://support.google.com/analytics/answer/2795983?hl=en.
[23] 2019. *Market share of Firebase.* Retrieved August, 2019 from https://www.appbrain.com/stats/libraries/details/firebase/firebase
[24] 2019. *Market share of Flurrt.* Retrieved August, 2019 from https://www.appbrain.com/stats/libraries/details/flurry/flurry-analytics
[25] 2019. *Market share of Google Analytics.* Retrieved August, 2019 from https://www.appbrain.com/stats/libraries/details/analytics/google-analytics
[26] 2019. *Market share of Mixpanel.* Retrieved August, 2019 from https://www.appbrain.com/stats/libraries/details/mixpanel/mixpanel
[27] 2019. *Privacy policy of emojidom.* Retrieved August, 2019 from http://www.emojidom.com/privacy-policy
[28] 2019. *Privacy policy of Staples.* Retrieved August, 2019 from https://www.staples.com/hc?id=dbb94c10-973c-478b-a078-00e58f66ba32
[29] 2019. *Privacy policy ofShopclues.* Retrieved August, 2019 from http://m.shopclues.com/rules-and-policies.html
[30] 2019. *UI/Application Exerciser Monkey.* Retrieved August, 2019 from https://developer.android.com/studio/test/monkey.html
[31] 2019. *Understanding PII in Google's contracts and policies.* Retrieved August, 2019 from https://support.google.com/analytics/answer/7686480?hl=en
[32] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.
[33] Mixpanel dashboard. 2018. . Retrieved October, 2018 from https://help.mixpanel.com/hc/en-us/articles/360000865566-Set-up-Your-Tracking/
[34] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.
[35] Seungyeop Han, Jaeyeon Jung, and David Wetherall. 2012. A study of third-party tracking by mobile apps in the wild. *Univ. Washington, Tech. Rep. UW-CSE-12-03-01* (2012).
[36] Jie Huang, Oliver Schranz, Sven Bugiel, and Michael Backes. 2017. The ART of App Compartmentalization: Compiler-based Library Privilege Separation on Stock Android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1037–1049.
[37] Xing Liu, Sencun Zhu, Wei Wang, and Jiqiang Liu. 2016. Alde: privacy risk analysis of analytics libraries in the android ecosystem. In *International Conference on Security and Privacy in Communication Systems*. Springer, 655–672.
[38] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. 2018. Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in mobile apps. In *Proceedings of the 2018 Annual Network and Distributed System Security Symposium (NDSS)(San Diego, California, USA.*

[39] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. 2014. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks.. In *NDSS*, Vol. 14. Citeseer, 1125.

[40] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. 2018. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. (2018).

[41] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. 2015. Haystack: In situ mobile traffic analysis in user space. *arXiv preprint arXiv:1510.01419* (2015), 1–13.

[42] Jingjing Ren, Martina Lindorfer, Daniel J. Dubois, Ashwin Rao, David R. Choffnes, and Narseo Vallina-Rodriguez. 2018. Bug Fixes, Improvements, ... and Privacy Leaks - A Longitudinal Study of PII Leaks Across Android App Versions. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*.

[43] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. 2016. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 361–374.

[44] Suranga Seneviratne, Harini Kolamunna, and Aruna Seneviratne. 2015. A measurement study of tracking in paid mobile applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 7.

[45] Jaebaek Seo, Daehyeok Kim, Donghyun Cho, Insik Shin, and Taesoo Kim. 2016. FLEXDROID: Enforcing In-App Privilege Separation in Android.. In *NDSS*.

[46] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. 2016. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 25–36.

[47] Connor Tumbleson and Ryszard WiÄŻniewski. 2017. Apktool-A tool for reverse engineering 3rd party, closed, binary Android apps.

[48] Narseo Vallina-Rodriguez, Jay Shah, Alessandro Finamore, Yan Grunenberger, Konstantina Papagiannaki, Hamed Haddadi, and Jon Crowcroft. 2012. Breaking for commercials: characterizing mobile advertising. In *Proceedings of the 2012 Internet Measurement Conference*. ACM, 343–356.

[49] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. 2018. Guileak: Tracing privacy policy claims on user input data for android applications. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, 37–47.

[50] Le Yu, Xiapu Luo, Xule Liu, and Tao Zhang. 2016. Can we trust the privacy policies of android apps?. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 538–549.

[51] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. 2013. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 611–622.

[52] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven Bellovin, and Joel Reidenberg. 2017. Automated analysis of privacy requirements for mobile apps. In *Proceedings 2017 Network and Distributed System Security Symposium*.