You Can't Always Get What You Want: Towards User-Controlled Privacy on Android

Davide Caputo, Francesco Pagano, Giovanni Bottino, Luca Verderame, and Alessio Merlo

Abstract—Mobile applications (hereafter, apps) collect a plethora of information regarding the user behavior and his device through third-party analytics libraries. However, the collection and usage of such data raised several privacy concerns, mainly because the end-user - i.e., the actual owner of the data - is out of the loop in this collection process. Also, the existing privacy-enhanced solutions that emerged in the last years follow an "all or nothing" approach, leaving the user the sole option to accept or completely deny access to privacy-related data. This work has the two-fold objective of assessing the privacy impact of mobile analytics libraries and proposing a data anonymization methodology that offers a trade-off between the utility and privacy of the collected data and enables complete control over the sharing process. To achieve that, we present an empirical privacy assessment on the analytics libraries used in the 4500 most-used Android apps of the Google Play Store in late 2020. Then, we propose an empowered anonymization methodology, based on MobHide (Caputo *et al.*, 2020), that gives the end-user complete control over the collection and anonymization process. Finally, we empirically demonstrate the applicability and effectiveness of our solution thanks to HideDroid, a fully-fledged anonymization app for the Android ecosystem.

Index Terms—Android privacy, analytics libraries, data anonymization

1 Introduction

The high number of mobile apps currently available for Android (nearly $4.83M$ in mid-2020 [2]) forced developers and companies to increase the quality of their apps in order to emerge in a fiercely competing market.

Users tend to choose the app to install according to both the number of its features and the ratings provided by customers [3], [4].

Thus, apps aim to maximize the user experience and tailor content to satisfy the users' expectations. Such a process forced developers to collect data about the users and their interaction with the apps, in order to evaluate their behavior and preferences, and enhance the app accordingly [5].

To this aim, analytics libraries allow developers to collect, filter, and analyze those data programmatically. They are typically composed of two elements: a client (i.e., an SDK), included in the app, and a back-end service. The SDK is responsible for collecting information regarding the device, the user, and her interaction with the app. The collected data is packed in data structures called *events* and sent to the back-end service through the network. The back-end, hosted as a cloud service, aggregates the received events and provides a dashboard for developers to analyze and filter the data.

• *The authors are with the University of Genova, 16146 Genova, Italy. E-mail: {davide.caputo, francesco.pagano, luca.verderame, alessio.merlo}@dibris.unige.it, giovannibottino3@gmail.com.*

Manuscript received 4 June 2021; revised 8 Nov. 2021; accepted 12 Jan. 2022. Date of publication 26 Jan. 2022; date of current version 14 Mar. 2023.
This work was partially supported in part by UNIGE Starting Grant Project "User-defined Data Privacy in Android" (2018).
(Corresponding author: Alessio Merlo.)
Digital Object Identifier no. 10.1109/TDSC.2022.3146020

The ease of use of analytics libraries, such as *Facebook Analytics* and *Google Firebase Analytics*, attracted a wide range of developers, leading to their rapid and widespread diffusion in the most popular mobile apps [6]. However, the adoption of such libraries raised several concerns on the privacy of the collected information, as described in [7] and [5]. For instance, several works (e.g., [8], [9]) reported how analytic libraries share the same privileges and resources of the hosting app and are able to access and collect sensitive information regarding the users and their behaviors without proper privacy-preserving mechanisms. Furthermore, authors in [10] demonstrated that only a negligible part of apps fulfills the Google Play privacy requirements for Android apps (1% out of the 5473 most downloaded apps in 2020).

The privacy issues of analytics libraries attracted the research community, which proposed several solutions to enhance the privacy of the collected data through anonymization techniques. For instance, Zhang *et al.* [11] proposed a solution allowing the developer to anonymize the collected information according to differential privacy techniques, while Liu *et al.* [5] designed an Android app able to intercept and block all the API related to analytics libraries. Also, Razaghpanah *et al.* [12] developed an app to block the network requests containing personal information.

Unfortunately, state-of-the-art solutions have some limitations. First, they do not give any control on the collected data and the anonymization process to the end-user, which is the actual data owner. In such a scenario, the anonymization solutions either autonomously select the type of data to collect and anonymize, or leave this choice to the app developer. Also, the proposed anonymization solutions follow an "all or nothing" approach, giving the sole option to either fully accept or deny the collection of personal data. Thus, the app developer can access the complete set of non-anonymized

data (100% utility of data, 0% privacy) or cannot access any information at all (0% utility, 100% privacy). Finally, existing solutions require invasive technical requirements such as adopting a customized OS, executing on a rooted device, the prior knowledge of personal data, or they require to customize the app logic. As a consequence, they could hardly be adopted in the wild.

Contributions of the Paper. In this work, we seek to address the following research questions (RQs):

- *RQ1*: *How widespread is the adoption of analytics libraries in mobile apps? Which are the most used libraries?*
- *RQ2*: *What is the impact of analytics libraries on the overall network traffic generated by the apps?*
- *RQ3*: *Which pieces of information are collected by analytics libraries, and how are they relevant for the users' privacy?*
- *RQ4*: *Is it possible to apply a local anonymization strategy compatible with existing analytics libraries that may grant the user a fine-grained control over the privacy of her data?*

We conducted an extensive experimental campaign over the first 4500 most downloaded Android apps from the Google Play Store between November 2020 and January 2021. We analyzed each app both statically and dynamically to evaluate the impact on privacy caused by the use of analytics libraries. Furthermore, we classified each collected data using the concepts of *Explicit Identifiers*, *Quasi Identifiers*, and *Sensitive Data* [13] to evaluate the privacy impact of the data collection process. Then, we empowered the MobHide methodology, proposed in our previous work [1], and we developed HideDroid (publicly available on GitHub [14]), a full-fledged anonymization app for Android. Finally, we tested HideDroid against the 4500 apps of the experimental testbed to assess the efficacy and applicability of local anonymization strategies according to the user's preferences. As an additional contribution, we released the entire dataset of anonymized network requests generated during the experimental campaign [15].

Structure of the Paper. The rest of the paper is organized as follows: Section 2 introduces the functionalities of analytics libraries and some basic concepts on data anonymization, while Section 3 presents the in-the-wild privacy analysis of the usage of analytics libraries in mobile apps. Section 4 details the MobHide methodology, describes the HideDroid anonymization app, and presents the evaluation of the usability and effectiveness of our solution. Section 5 discusses the current state-of-the-art, while Section 6 concludes the paper and points out some future extensions of this work.

2 Background

2.1 Analytics Libraries

Analytics libraries are software solutions that allow developers to monitor and track the user's interactions with their apps. They are typically composed of two parts, namely the client library and the back-end system. The client library consists of a Software Developer Kit (SDK), containing a set of APIs for the in-app data collection and the auxiliary scripts to include and compile the client inside the app package. The back-end system, either available as a cloud service or an on-premise solution, is responsible for collecting and aggregating the clients' data and gives the developer a full-featured dashboard to review, analyze, and extract the requested information.

Analytics libraries enable collecting a wide range of information belonging to two macro-categories: *personal data* and *event data* [16], [17], [18]. Personal data include details about the user, such as the email address or the username, and the device, e.g., device name, SDK version, and the network carrier. Event data focus on the interactions between the user and the app. Analytics libraries provide to developers *i)* a set of predefined events, like *"app open"* or *"app close"*, and *ii)* the possibility to create custom events. Examples of custom events include: *"add to cart"*, *"add payment info"*, and *"purchase"* in case of e-commerce apps, or *"click to Ad"*, *"rewarded video"* in case of mobile games.

Personal and event data are encoded in key-value data structures and sent by the client library to the back-end system using the network connectivity, e.g., through HTTPS connections.

2.2 Anonymization Techniques

In data privacy, the set of attributes in a microdata set [13], [19] can be mainly divided into three categories:

- *Explicit Identifiers (EI)*. EI are user-identifying attributes, such as the name/surname, the social security number (SSN), or the Insurance ID.
- *Quasi-Identifiers (QI)*. This category includes attributes that can be combined with other external data sources (e.g., publicly available databases) to indirectly identify a user. Examples of QI include geographic and demographic information, phone numbers, and e-mail IDs.
- *Sensitive Data (SD)*. SD are attributes that contain relevant information for the recipient of the microdata set, like, e.g., health diseases, salaries, eating habits, just to cite a few.

Data Anonymization (DA) techniques aim to decouple the user's identity (i.e., EI and QI) from her sensitive information (i.e., SD) before releasing the microdata set in the wild. To do so, DA techniques first remove or substitute the EI and then alter the QI set to reduce the possibility to re-identify the user through external databases, and then correlate her identity with the corresponding SD attributes.

DA techniques can be divided into two groups: *Perturbative (PT)* and *Not Perturbative (NPT)*. *PT* techniques consist of altering the QI data with dummy information to weaken their correlation. For instance, a numeric attribute, e.g., the *zip_code = 16011*, can be transformed to *zip_code = 16129* by adding a noise equal to 118. *NPT* techniques aim at reducing the detail in the data through generalization of values, with a very limited impact on the semantics of data. As an example, the value *zip_code = 16011* can be generalized to *zip_code* $= 160**$.

The application of each anonymization technique can be evaluated according to the level of *privacy* and *utility* of the processed data. The two values are inversely proportional: the more anonymization is applied, the higher level of user's privacy is granted (i.e., the probability to de-anonymize the user is low), at the expense of the utility of the data

(i.e., the semantics data is highly affected). Conversely, if the level of anonymization is low, the utility of data is high, but the level of user's privacy is reduced, thus raising the probability to de-anonymize a user in the released microdata set [19], [20].

Unfortunately, PT techniques have several limitations in terms of utility and privacy. Indeed, complex noise transformations severely alter the semantics of data resulting in a significant utility loss, as described in [13], [20]. On the other hand, simple noise distortion techniques can be reverted to obtain the original microdata set, as demonstrated by [13], [21], [22].

In our work, we focused mainly on two NPT anonymization techniques: *Data Generalization* (DG) [23] and *Differential Privacy* (DP)[19].

Data Generalization. DG replaces specific values of a set of attributes belonging to the same domain, with more generic ones [24]. In a nutshell, given an attribute A belonging to a domain $D_0(A)$, it is possible to define a *Domain Generalization Hierarchy (DGH)* for a Domain (D), as a set of n anonymization functions f_h: $h = 0, ..., n-1$, such that

$$D_0 \xrightarrow{f}_0 D_1 \xrightarrow{f}_1 \ldots \xrightarrow{f}_n -1 D_n, \tag{1}$$

and

$$D_0(A) \subseteq D_1(A) \subseteq \ldots \subseteq D_n(A). \tag{2}$$

It is worth noticing that the more generalization functions are invoked on the original data, the higher is the resulting privacy value (and the lower is the data utility), as heterogeneous data are transformed into a more reduced set of general values. Generalization techniques are suitable for semantically independent data, such as the set of *personal data* collected by analytics libraries (e.g., the *deviceName* or the *phone number*).

Differential Privacy. The DP technique consists of altering the original distribution of a set of interdependent data using a perturbation function [25]. This approach is usually applied in a context where i) the main requirement is the confidentiality of the data exchanged between pairs, and ii) the receivers' identity is unknown a priori. There are two main models for defining DP problems: *centralized* and *local* model. In the centralized model, the data are sent to a trusted entity that applies DP algorithms and then shares the anonymized dataset with an untrusted third-party client [25]. On the contrary, the local model assumes all external entities and communication channels as untrusted [26], [27]. In such a situation, local DP techniques aim at performing the data perturbation locally before releasing any dataset to an external party.

In our scenario, we consider the user as the sole owner of its data, and we trust neither the analytics company nor the developer. To this aim, the local DP model is suitable to anonymize sequences of events logged by analytics libraries. The objective of DP is to transform a sequence of events $(e_1, e_2, \ldots, e_n) \in D$ in a different sequence of events $(z_1, z_2, \ldots, z_n) \in D$ through the application of a perturbation function $R : D \rightarrow D$ to each event. This function is commonly a probability distribution, defined a priori: $z_i = R(e_i)$.

TABLE 1
Distribution of Analytics Libraries in the Top 4500 Android Apps

Analytics Library	# App	Percentage
Google Firebase Analytics	3569	79.3%
Google AdMob	3371	74.9%
Google CrashLytics	2093	46.5%
Facebook Login	1688	37.5%
Facebook Ads	1616	35.9%
Facebook Share	1580	35.1%
Facebook Analytics	1517	33.7%
Unity 3d Ads	1244	27.2%
Google Analytics	1103	24.5%
Moat	1008	22.4%
Google Tag Manager	873	19.4%
AppLovin	842	18.7%
Facebook Places	806	17.9%
ironSource	801	17.8%

3 Analytics Libraries In The Wild

In this section, we evaluate the presence of mobile analytics libraries in Android apps and their impact on the security of the device and the user (RQ1-RQ3) by conducting an experimental campaign on the 4500 most downloaded Android apps taken from Google Play Store between November 2020 and January 2021.

Despite there exist some other works that analyze the spread of third-party libraries on mobile apps (e.g., [5], [28]), such proposals have some important limitations: i) they investigate the network traffic of third-party libraries for specific categories of apps only (e.g., parental control apps [29], paid apps [30] or pre-installed apps [31]), ii) the analysis does not inspect the content of network requests and its privacy impacts, and iii) they do not share a public dataset that could be used for future research activities. Those considerations drove us to present a new analysis with a specific focus on mobile analytics libraries that allowed for the creation of an updated dataset that we publicly released to the research community [15].

To address *RQ1*, we statically analyzed all the downloaded apps to identify the presence of analytics libraries in the app code. Each Android app has been scanned using *Androguard* [32] and *Exodus Core* [33] to detect if it includes package names belonging to analytics libraries (e.g., the package *com.google.firebase.crashlytics* refers to the use of Google CrashLytics library).

Table 1 summarizes the most included libraries in the dataset. It is worth noticing that the most used analytics libraries belong to Google (92.5%) and Facebook (52.8%).

The analysis of the impact of analytics libraries on the network traffic generated by apps (*RQ2*) as well as on the privacy of the user and the device (*RQ3*) required a dynamic analysis phase to evaluate the behavior of the different analytics solutions *at runtime*. Also, dynamic analysis allowed detecting also obfuscated or runtime-loaded libraries that can hardly be detected through static analysis.

To do so, we tested each app for 10 minutes to collect the generated network traffic. Apps were tested using DroidBot [34], a black-box framework that automatically stimulates the app under test (AUT) by mimicking human interactions. The testing environment consists of an emulated Android

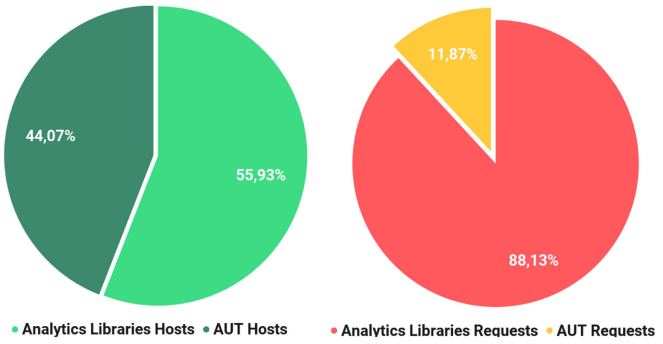


Fig. 1. Impact of analytics libraries on the network traffic in terms of (a) contacted hosts and (b) generated network requests.

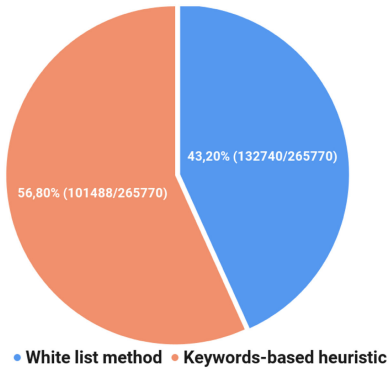


Fig. 2. Impact of the two classification methods in the identification of network requests of analytics libraries.

device with Android OS version 10 and root permissions. For the traffic collection, we relied on *mitmproxy* [35]. This tool enables the deep inspection of SSL connections thanks to the installation of a custom CA certificate in the system certificate directory (allowed by root permissions). To further cope with apps and libraries implementing SSL Pinning techniques [36], [37] to protect the network traffic, we dynamically instrumented each AUT using Frida [38] in order to bypass the most common implementations of SSL pinning.

The dynamic analysis has been carried out in an Ubuntu 20.04 VM with 32 GB of RAM and 16 cores at 3.8 GHz. To speed up the evaluation phase, we used three Android emulator instances at a time. The analysis lasted 14 days and collected 265770 unique network requests generated by the AUTs.

The collected traffic has been inspected to determine if it belongs to an analytics library. In detail, we classified all the network traffic according to a list of well-known network hosts connected to analytics libraries (extracted through the Exodus tool [33]).

Moreover, we proposed a heuristic based on the parsing of the network requests according to a set of keywords (e.g., *device-name*, *device-id*, *device-info*, *event-type*, *event-info*, *event* or *event-name*), to cope with the possible presence of unknown hosts. Such keywords are typically included in events generated by analytics libraries [1]. If the request contains at least one of them, the heuristic labels the request and keeps track of the new host. Finally, any host identified using the heuristic has been manually inspected to detect and remove false positives.

Fig. 1a details the impact of analytics libraries in terms of contacted hosts (a) and number of network requests (b). In particular, among 1482 unique network hosts, only 653 (i.e., the 44.07%) are related to the normal behavior of the AUTs, while the remaining 829 (i.e., the 55.93%), are connected to an analytic service (Fig. 1a). Furthermore, it is worth noticing that over 88.13% of the resulting requests (i.e., 234228 out of 265770) are related to analytics services (Fig. 1b), confirming that analytics frameworks have a significant impact on the overall network traffic of apps (*RQ2*).

Fig. 2 depicts the contribution of our heuristic in the identification of network requests associated with analytics services with respect to the traditional white-list methodology. In detail, the experimental campaign allowed the identification of 132.808 new requests, representing 56.70% of the total, and the mapping of 576 new hosts with the corresponding service.

Moreover, we identified the top 20 analytics hosts contacted during the dynamic analysis phase to further confirm the results obtained through the static analysis. Indeed, the 77% of network requests (i.e., 181804) belong to *firebaseinstallations.googleapis.com* (i.e., Google) and *graph.facebook.com* (i.e., Facebook) (see Fig. 3).

Concerning the events stored by these libraries, Fig. 4 reports the set of events recorded during the dynamic analysis phase, sorted in decreasing order of frequency. The most frequent event is *"CUSTOM_APP_EVENTS"*, belonging to the *graph.facebook.com* analytics service. Through this attribute, developers can define custom events.

The evaluation of the privacy impact of analytics libraries (*RQ3*) required an in-depth review of the content of the network requests. For each request, we extracted all the attribute keys of the event. Then, we ordered the list of attributes according to the frequency of appearance, and we classified them by evaluating their privacy impact (i.e., EI, QID, SD). The analysis of the dataset generated by the dynamic analysis phase led to the extraction of 6025 unique attributes. Each attribute has been manually evaluated according to 1) its content, 2) the ability to identify the user, and 3) its role inside the whole event.

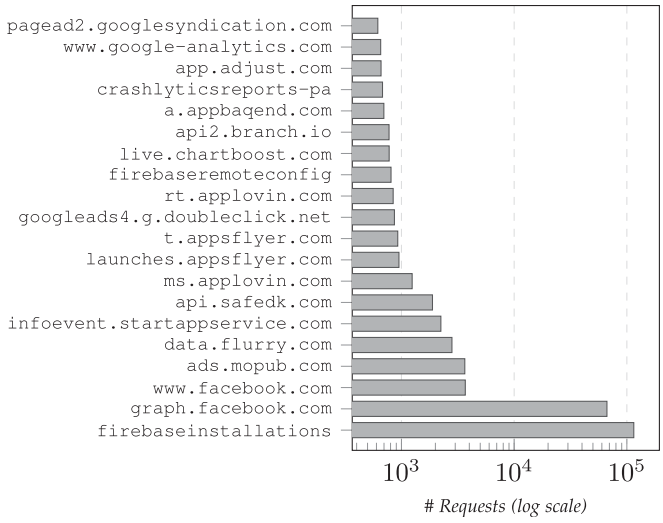


Fig. 3. Distribution of the network requests related to analytics libraries.

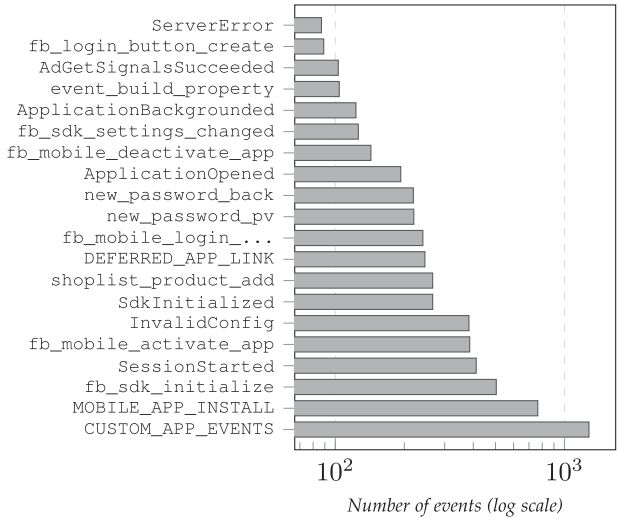


Fig. 4. Distribution of analytics events recorded during the dynamic analysis phase.

Table 3 lists the 44 most used attributes. For each entry, we describe the attribute, an example of its usage, the privacy classification (i.e., EI, QID, SD), the number of occurrences, and the number of network hosts that received such an attribute.

The extraction and classification phase highlighted the presence of:

- a set of EI that enables the unique identification of app installations (e.g., *appID* and *fid*), users (e.g., *uid* and *uuid*), or devices (e.g., *device_id* and *hardware_id*);
- a set of QI that may allow for the indirect identification of the user. Examples of QI include *location*, *SSID Device Name*, and *Device Manufacturer*;
- a set of SD related to the events generated by the interaction with the AUT that allow developers to infer the user behavior. Those attributes include *name*, *type*, *duration*, *events*, *event_type*, and *event*.

Table 2 points out the percentage of how often a request sends a data classified as privacy relevant (see Table 3) to the most used analytics hosts. It is worth pointing out that each request contains at least one EI, as well as that all requests expose - without any form of anonymization - at least one QID.

Overall, the privacy analysis on the information extracted by analytics libraries allowed us to empirically demonstrate their relevance for the user's privacy. Also, it is worth noticing that the experimental results provide no evidence that analytics services apply any anonymization or application-level encryption on the collected data. As a consequence, there is the need for a viable and scalable anonymization strategy that grants a fine-grained user control over personal data, thereby ensuring compatibility with the existing mobile analytics solutions.

4 MobHide Local Anonymization

The need to determine the feasibility of applying user-centric anonymization techniques to the information collected by analytics libraries (*RQ4*) drove us to extend our previous work on local data anonymization on mobile devices [1]. In detail, we refined the *MobHide* per-app anonymization methodology to cope with the state-of-the-art mobile analytics frameworks. Furthermore, we extended the *HideDroid* prototype for the Android ecosystem to perform extensive and in-the-wild analysis on real Android apps. The rest of this section summarizes the MobHide methodology, presents the extension of the *HideDroid* prototype, and discusses the results of the experimental activity on the dataset of 4500 Android apps.

TABLE 2
Percentage of EI, QID, SD for Each Request in the Most Widespread Hosts

Host	EI %	QID %	SD %	# Req
firebaseinstallations.googleapis.com	99%	100%	0.1%	115217
graph.facebook.com	1.9%	100%	3%	66587
www.facebook.com	95%	100%	95%	3685
ads.mopub.com	97%	100%	6.4%	3648
firebaseremoteconfig.googleapis.com	41%	100%	1.3%	2805
rt.applovin.com	0.04%	100%	0%	2237
googleads4.g.doubleclick.net	0.1%	100%	2%	1885
t.appsflyer.com	100%	100%	100%	1244
launches.appsflyer.com	0.6%	100%	0.1%	950
ms.applovin.com	1.6%	100%	0%	927

4.1 MobHide

The MobHide methodology allows the user to choose a different privacy level for any app installed on the device. The idea is to dynamically analyze the app behavior at runtime and anonymize the data exported by analytics libraries. In detail, MobHide leverages runtime monitoring of any app according to the following steps: i) intercept all the events generated by the analytics libraries, ii) anonymize the information therein by applying generalization and local DP techniques, and iii) send the anonymized data to the backend by mimicking the original network calls.

However, the analysis on analytics libraries discussed in Section 3 led us to revise and extend the methodology (and the prototype) originally proposed in [1].

First, the previous methodology relied on a predefined list of well-known hosts of the analytics services. However, the difficulty in maintaining an updated list led to several false negatives in the preliminary experimental results.

Thus, we revised the MobHide methodology by: i) extending the Privacy Detector Module with a keyword-based heuristic (c.f. Section 3) and ii) including a dedicated support element, called *Analytics Domain DB*, that can dynamically update the mapping among the hostnames and the corresponding analytics services.

Also, the previous proposal envisaged the storage of the intercepted network requests in a buffer, carrying out the local DP anonymization and subsequently their forwarding in blocks of n requests. Thus, the original HideDroid prototype needed to intercept and hang a pool of open connections to analytics services to reach the desired block of requests to trigger the anonymization process. Unfortunately, the preliminary experimental evaluation showed that such behavior is hardly achievable in a real scenario. In detail, HideDroid was unable to collect a sufficient number of events before the poll of connections expires due to protocol timeouts. Furthermore, the idea of dropping the original

TABLE 3
Evaluation of the Most Privacy-Relevant Attributes Collected by Analytics Libraries

Keyword	Description	Example	Type	# Occ	# Hosts	Keyword	Description	Example	Type	# Occ	# Hosts
appID	Identifies a specific installation of an app, mainly used by Google libraries	"appId": " 1:344560015735:android:482ed113bf00cb81"	EI	118321	1223	*uid*	User Identifier	"uid":"1609497426061-1356139383227225046"	EI	2064	10
fid	FirebaseID, identifies a specific installation of an app	"fid":"cUSx9iNuTY2u8xAUm-9tkA2"	EI	115211	1045	*event_type*	Specific event, generated by a user	"event_type" : "ClickTopProductCard"	SD	2046	23
name	Specific event, generated by a user	"name" = "fb_app_events_enabled"	SD	26912	288	*user*	Information about user and device	"user":{ "deviceId" : "66dfda9e-xxxx-xxxx-xxxx-fd4909f19131", ...}	EI	1923	147
type	Used to identify a specific event	"type": "perf.startupTime.v1"	SD	11262	329	*udid*	User Identifier, used by mopub	"udid" :"mopub:ca802221-f769-4909-aef7-46c80be7353b"	EI	1889	127
model	Specifies the device used	"model": "Android SDK built for x86"	QID	11051	486	*device_id*	Device Unique Identifier	"deviceId":"ffffffff-b626-4582-a9f2-20d36d7a4fe6"	EI	1887	125
device	Device name	"device": "generic_x86"	QID	7548	423	*android_id*	ANDROID_ID	"android_id": "5439903757 9f251d"	EI	1662	69
locale	Device language	"locale": "en_US"	QID	6166	319	*operator*	Mobile Telephone Operator	"operator": "T-Mobile"	QID	1610	81
manufacturer	Device manufacturer	"manufacturer":"Google"	QID	6141	261	*userId*	Identifier associated with the device	"userid" : "02903474b885d424d9-a61ea307dc850a"	EI	1502	85
os	OS Version and related Information	"os": {"version":"10", "buildVersion":"REL", ...}	QID	5230	379	*location*	Identifies the location where the event has been generated	"location": "Home Screen"	SD	1492	55
carrier	Information about OS and Mobile TelephoneOperatore	"carrier": {"carrier-name" : "Android", "iso-country-code":"us",...}	QID	5023	319	*deviceData*	Information about device hardware	"deviceData": {"cpu_abi" : "x86", ...}	EI	1476	61
current	ID of the current event	"current":"15"	QID	4683	9	*uuid*	Unique identifier generated by analytics services, used to identify a device	"uuid": "2bdc76f3-63d1-4184-93bd-060918ae6bea"	EI	1171	133
timezone	Timezone	"timezone" : "Asia/Kabul"	QID	4645	261	*appInstanceId*	App Instance	"appInstanceId": "dVjk1CTB9kU"	QID	1161	252
app	Identify a specific app installation. Used by Google libraries	"app": { "installationUuid":"cad471b7-2b064bd4b29e5d8120161f94",... }	QID	4427	237	*ad_id*	Google Advertiser Id	"ad_id": "66dfda9e-9236-459f-9d9c-fd4909f19131"	EI	1160	100
language	Device Lamguage	"language"="en"	QID	4152	367	*installationUuid*	Hash relativo ad un'applicazione installata sul dispositivo	"installationUuid": "4882882d9829459e89d860c59..."	QID	1002	97
country	Country	"country":"IT"	QID	4089	234	*hardware_id*	ANDROID_ID	"hardware_id":"033ae95da00-85566"	EI	903	64
carrierName	OS & Mobile Telephone Operator	"carrierName" = "Android"	QID	3562	71	*identity*	Base 64 of uuid and gaid	"identity" : "eyJ1dWlkIjoiMTc1Zj-Y2OWMwYjVhYjc2ZCIsImdh..."	EI	834	24
data	Additional Information about an event	"data":{" requests_count ":32, "events_count ":91, "attributes_count":18 }	QID	3371	152	*bssid*	Mac Address of AP	"bssid": "02:00:00:00:00:00"	EI	738	15
advertiserId	Advertiser User Id	"advertiserId" : "8e83d747-xxxx-xxxx-xxxx-..."	EI	3353	107	*event*	Event Name	"event": MOBILE_APP_INSTALL	SD	675	45
duration	Event duration	"duration" : 11270	SD	2945	59	*deviceIP*	Device IP	"deviceIP":"xxx.0.2.xx"	EI	671	6
mid	Mopub Generated Identifier	"mid":"8bfbff85-xxxx-xxxx-xxxx-a488bae44e1f"	EI	2931	100	*ssid*	SSID name	"ssid":"<unknown ssid>"	QID	622	12
events	List of information associated to a specific events	events: {"type": "deviceInfo", "ts" : 1607083813320, "os_ver" : "Android OS 10 API-29", ...	SD	2656	296	*mac_address*	Device Mac Address	mac_address="yy:xx:zz:yy:xx:-yy"	EI	506	37
network	Network Informaion	"network": "MOBILE"	QID	2568	115	*deviceFingerPrintId*	Device Fingerprint	"deviceFingerPrintId": "ffffffff-bf43-71d1-ffff-ffffef05ac4a"	EI	505	22

connections and starting new ones after the anonymization process was ineffective since the analytics clients within the app tried to re-connect and re-send the same set of events, resulting in an erroneous behavior and, in some cases, to the crash of the app.

To overcome this problem, we also revised the MobHide anonymization pipeline, where each request is intercepted, anonymized, and forwarded sequentially. In detail, the new methodology exploits the local DP technique by defining a perturbation function R (see Section 2.2) on the history of previous events for the same hostname rather than on a block of new intercepted events.

Fig. 5 provides a high-level view of the new MobHide workflow. The *Privacy Detector* module (step 1) intercepts the network requests generated by the user apps and detects those referring to analytics services exploiting a list of well-known network hosts connected to analytics libraries and the heuristics described in Section 3. If the network request does not belong to an analytics library, the module transparently forwards it to the destination server (step 5). Otherwise, the *Privacy Detector* stores the request in the Event Buffer module (step 2), sends the data within the request to the *Data Anonymizer* (step 3), and updates the domain name in the *Analytics Domain DB* (step 4). The *Data Anonymizer* is the core module of the MobHide methodology and it is responsible for the anonymization task. The complete procedure implemented in the *Data Anonymizer* is described in Algorithm 1.

The algorithm takes as input the intercepted request (i.e., $currReq$), the privacy level chosen by the user (i.e., $currPL$), the minimum number of requests (i.e., $minLen$) to carry out the DP anonymization, and the data stored in the event Buffer. For each request intercepted by the *Privacy Detector*, the algorithm extracts the package name of the app (i.e., $appName$) and the destination server (i.e., $hostName$) (rows 1-2). Using these values, the algorithm extracts all requests between these two entities from the $eventBuffer$ (row 3).

Then, the module initializes the output list of anonymized requests (i.e., $anonymizedRequests$) (row 4) and the threshold value (i.e., $Threshold_{action}$) (row 5), defined as

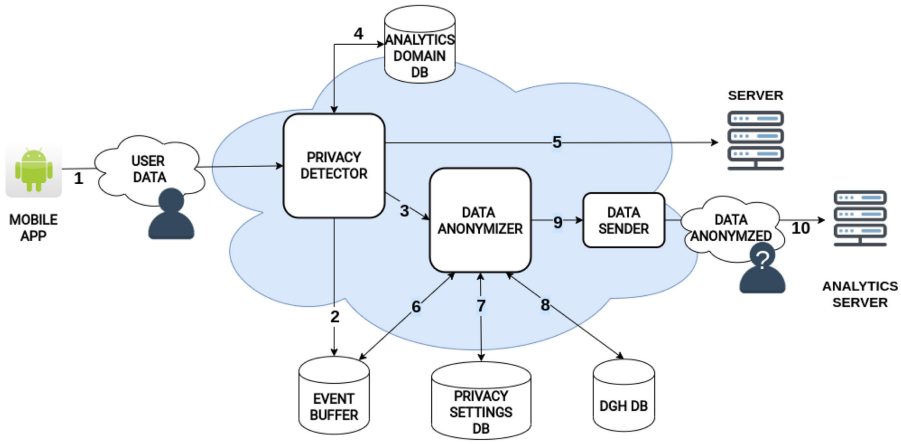


Fig. 5. MobHide methodology.

$$Threshold_{action} = 1 - \frac{currPL}{\#action + 1}, \quad (3)$$

where

$$action \in [\texttt{inject}, \texttt{remove}, \texttt{replace}].$$

$Threshold_{action}$ is used to determine the possible actions of the perturbation function R (i.e., *inject*, *remove* or *replace* as defined in [1]).

Algorithm 1. Data Anonymization Pipeline

Input: $currReq$, $currPL$, $eventBuffer$, $minLen$
Output: $anonymizedRequests$

1: $appName \leftarrow currReq$.appName
2: $hostName \leftarrow currReq$.hostName
3: $history \leftarrow eventBuffer$.extractReq($appName, hostName$)
4: Initialize $anonymizedRequests \leftarrow list()$
5: Initialize $Threshold_{action} \leftarrow 1 - (currPL/4)$
6: $currReq.att \leftarrow$ generalizeData($currReq.att, currPL, history$)
7: **if** len($history$) $\geq minLen$ **then**
8: $Pr_{inj} \leftarrow$ rand()
9: $Pr_{rem} \leftarrow$ rand()
10: $Pr_{rep} \leftarrow$ rand()
11: **if** $Pr_{inj} > Threshold_{action}$ **then**
12: $newReq \leftarrow$ genNewRequest($currPL, history$)
13: $anonymizedRequests$.add($newReq$)
14: $anonymizedRequest$.add($currReq$)
15: **end if**
16: **if** $Pr_{rep} > Threshold_{action}$ **then**
17: $replReq \leftarrow$ replaceRequest($currReq$)
18: $replReq.att \leftarrow$ generalizeData($replReq.att, currPL, history$)
19: $anonymizedRequests$.add($replReq$)
20: **else if** $Pr_{rem} > Threshold_{action}$ **then**
21: deleteEvent($currReq$)
22: **else**
23: $anonymizedRequest$.add($currReq$)
24: **end if**
25: **else**
26: $anonymizedRequest$.add($currReq$)
27: **end if**
28: **return** $anonymizedRequests$

The next step consists in the generalization process of the original request, i.e., $generalizeData$ (row 6). Then, if the history of requests between two endpoints is above $minLen$ (row 7), the *Data Anonymizer* applies the local DP anonymization and computes the three pseudo-random numbers, i.e., $Pr_{inj}, Pr_{rem}, Pr_{rep}$, used by the perturbation function R to inject, remove or replace the event, respectively (rows 8-10). If Pr_{inj} is higher than the threshold (row 11), the *Data Anonymizer* module picks a random generalized event from the history (row 12) and adds the new request to the $anonymizedRequests$ list (row 13) along with $currReq$. If Pr_{rep} is greater than the threshold (row 16), the module replaces the original event with one extracted from the history, i.e., $replReq$ (row 17), generalizes it (following the rules described in [1] and using the information stored in the *DGH DB*, step 8) (row 18) and adds $replReq$ to the $anonymizedRequests$ list (row 19). In case Pr_{rem} is greater than the threshold (row 20), the *Data Anonymizer* module removes the original request (row 21).

Once the request has been anonymized, the *Data Anonymizer* forwards the $anonymizedRequests$ to the *Data Sender* (step 9). The *Data Sender* assembles the new anonymized network request and forwards it to the analytics backend (step 10).

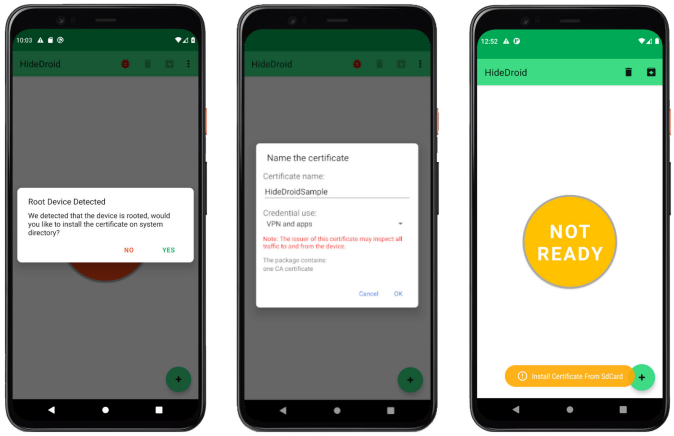
4.2 HideDroid

HideDroid implements the MobHide methodology for the Android ecosystem as a user app compatible with Android 6.0 and above. The application, after an initial configuration (*Initial Setup*), enables users to select a privacy level for each of the installed apps (*Per-App Privacy Configuration*) and, thanks to an embedded network proxy, allows the traffic collection and anonymization phase (*Runtime Anonymization*). HideDroid is publicly available on GitHub [14].

4.2.1 Initial Setup

HideDroid requires an initial configuration to successfully intercept the network traffic generated by the apps. At the first execution, HideDroid requires the permission to access the external storage (i.e., `WRITE_EXTERNAL_STORAGE`) to store the intercepted network traffic.

Then, the app generates and installs a self-signed certificate for the network traffic collection. During such a process, HideDroid checks if the device has root permissions. If this is the case, the app requests the permission to install the



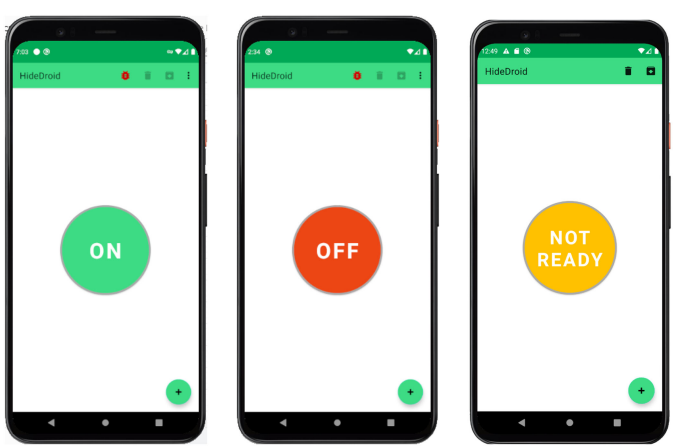
(a) (b) (c)

Fig. 6. HideDroid prompts. *(a)* Root detection. *(b)* In-app certificate installation (Android 10 or lower). *(c)* Manual installation of the CA certificate (Android 11+).

certificate within the system directory, which is inaccessible with the default user permissions (Fig. 6a) [39]. The certificate installation in the system directory allows HideDroid to bypass an extra configuration step in the next phase (i.e., the *repackaging phase*). In case the device has default permissions only, HideDroid executes two different actions based on the Android version installed on the device. If the Android version is lower than Android 11, HideDroid prompts the user to install the proxy certificate in the user directory (Fig. 6b). Otherwise, the app asks the user to install the certificate (Fig. 6c) manually. Such action is needed because Android 11 and above have tightened the restrictions on CA certificates, denying any app, debugging tool, or automated action to prompt the installation of a CA certificate [40].

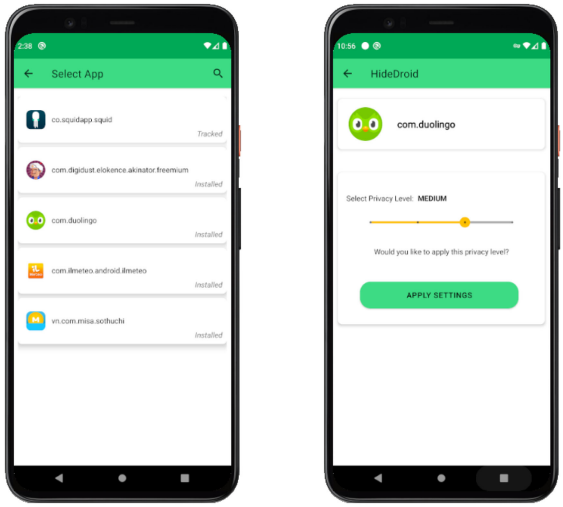
4.2.2 Per-App Privacy Configuration

Once the setup phase has completed, HideDroid displays the home page screen (Figs. 7a and 7b). Here, the user can activate the anonymization mode (on/off button) and select the apps to shield (plus button). If this is not the case, HideDroid displays a Not Ready warning (Fig. 7c). This screen is



(a) (b) (c)

Fig. 7. HideDroid home screen. *(a)* Incognito Mode on. *(b)* Incognito Mode off. *(c)* Not Ready.



(a) (b)

Fig. 8. Per-app privacy configuration.

prompted until the certificate is successfully installed on the device.

The app selection process shows the list of all the installed apps (Fig. 8a). For each app, the user can select the desired privacy level through a slider widget (Fig. 8b).

If the HideDroid certificate is not installed in the system certificate store and the device has Android OS version $\geq$ 7.0, the tool requires the *repackaging* of each app whose selected privacy level is above `NONE`.

Such an additional step is mandatory to overcome the network security restriction imposed by the newer Android versions [41], [42] that do not recognize user certificates as trusted by default.

In detail, HideDroid automatically unpacks each selected app, overwrites the `network_security_config.xml` file (Listing 1) to include a new trust-anchor for user-defined certificates, rebuilds the apk files, and installs the modified versions.

It is worth noticing that - thanks to the repackaging phase - the user is not forced to have (and grant) root permissions to HideDroid, thereby ensuring a wider compatibility w.r.t. the state-of-the-art solutions. Also, the repackaging is only required for devices with OS $\geq$ 7.0 to support the newest Android OS versions without breaking their standard security model. Finally, the modification of the app neither alters the compiled code nor other resources of the apps.

Listing 1. Example of Network Configuration

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <!-- Trust preinstalled CAs -->
            <certificates src="system" />
            <!-- Additionally trust user added CAs -->
            <certificates src="user" />
        </trust-anchors>
    </base-config>
</network-security-config>
```

4.2.3 Runtime Anonymization

The *Data Anonymization* is the core phase of HideDroid workflow. In detail, HideDroid implements the MobHide

TABLE 4
Content Type and Content Encoding Supported by HideDroid

Content-Type	application/x-www-form-urlencoded
Content-Type	application/json
Content-Type	multipart/form-data
Content-Encoding	gzip
Content-Encoding	deflate

TABLE 5
Parsing Table of Analytics Network Requests

Headers	Intercepted request header
Content-Type	Intercepted request Content-Type
Content-Encoding	Intercepted request Content-Encoding
URL	Destination address (host and path) and request type (POST, GET, PUT, etc.)
Body	Intercepted request body
App	App Name

anonymization pipeline to anonymize the traffic generated by analytics libraries of all the apps configured in the previous phase with a privacy level LOW, MEDIUM, or HIGH. The core of the pipeline is implemented as two background services, i.e., the *Privacy Interceptor* and the *Data Anonymizer*.

The *Privacy Interceptor* component has a twofold objective. The first is to collect the network traffic of all apps with a selected privacy level above NONE. The second is to filter network requests that do not belong to any analytics services.

To intercept network requests generated by apps, the component relies on and exploits the Android VPN API [43]. These APIs allow building a transparent VPN that acts like a proxy server between the client (i.e., the app) and the server (i.e., the analytics server). This solution enables HideDroid to intercept the network traffic generated both from Java/Kotlin and native code, and it is able to inspect both HTTP and HTTPS traffic by exploiting SSL deep-inspection techniques.

Then, the Privacy Interceptor differentiates the traffic generated by apps according to their package names. For Android versions lower than 10, the module leverages the /proc/net file to obtain the UID of the app that generated a specific request. For Android versions above 10, the Privacy Interceptor uses the getConnectionOwnerUid[1] method [44]. Finally, the module maps the UID of the process with the package name of the app using the getPackagesForUid[2] API.

After the collection phase, the Privacy Interceptor filters all the collected network traffic to identify the one belonging to analytics services. At first, the module queries the *Analytics Domain DB* to spot well-known domain names belonging to analytics frameworks. If no match is found, then the service enforces the heuristic described in Section 3 to evaluate the request. If the domain is recognized as belonging to an analytics service, the module blocks the request and stores it for the anonymization phase. On the contrary, the network request is transparently forwarded to the destination.

The *Data Anonymizer* is responsible for applying the anonymization strategies on all the stored network requests. As a preliminary step, the service decodes each network request to preserve the original structure after the anonymization. For each request, the service extracts information regarding the headers and the body of the request, as shown in Table 5. The current version of HideDroid supports the Content-Types and Content-Encodings listed in Table 4. After the parsing phase, the Data Anonymizer can employ the anonymization process using the *Generalization* and *Differential Privacy* techniques following the Anonymization Pipeline of Algorithm 1. Finally, the anonymized request will be encoded in the original form and forwarded to the original target server by the *Data Sender* module.

4.3 Testing HideDroid In The Wild

We conducted another experimental campaign on the same dataset of 4500 Android apps used for the evaluation of analytics libraries in the wild (cf. Section 3), in order to evaluate the effectiveness and efficacy of HideDroid. The experiments relied on an emulator equipped with Android 10 without root permissions. By using such an environment, we were able to test all steps performed by HideDroid, including the repackaging phase, which is not mandatory in case of root permissions or Android OSes below 7.0 (see Section 4.2.1). We tested each app for 10 minutes, including the time to perform the configuration tasks (repackaging, privacy-level selection) before executing the app.

Runtime Performance.The experimental evaluation of 4500 apps lasted one month. HideDroid was able to successfully process and anonymize data belonging to 3992 apps (i.e., 88.7%). The remaining 508 apps (i.e., 11.3%) were not tracked by HideDroid due to the failure of the repackaging phase. The root-cause analysis of the failure allowed us to identify that the failure was triggered during the re-installation of the modified app. In detail, the error is generated by the *VerifyAdvancedProtectionInstallTask* method of the com.google.android.finsky process. Such control verifies the app signature by comparing it with the original one. If the two signatures do not match, the process blocks the installation. As additional proof, we also tested the AUTs that failed in an Android emulator with root permissions, confirming their actual functioning.

The dynamic testing phase allowed measuring the impact of the delays introduced by the anonymization pipeline on the AUTs. To do so, we measured the delay introduced by the interception, anonymization, and forward of each analytics event. HideDroid was able to process, on average, an event in 52.84 ms with a standard deviation of 122.18 ms, thus, confirming a negligible impact on the AUTs.

Furthermore, we also evaluated the compatibility of HideDroid for the leading analytics services during the experimental phase. In detail, we tracked the acceptance rate of the anonymized requests by the analytics back-end services. Table 6 shows the percentage of anonymized events accepted by the ten most used analytics services. On average, the acceptance rate of most of the analytics services like Google ADS and Facebook is above 93.69%. The only notable exception is the Firebase Analytics services that

1. https://github.com/Mobile-IoT-Security-Lab/HideDroid/blob/main/netbare-core/src/main/java/com/github/megatronking/netbare/net/UidDumper.java

2. https://developer.android.com/reference/android/content/pm/PackageManager#getPackagesForUid(int)

TABLE 6
Acceptance Rate of the Anonymized Events of the Top 10 Analytics Backend Services

Service	Acceptance Rate
Firebase Analytics	0.19%(317/164272)
Facebook Audience	93.69%(81394/86872)
Google DoubleClick	99.46%(85516/85982)
Google AdMob	99.96%(47713/47733)
Google Tag Manager	99.88%(6840/6848)
Google Ads	93.94%(5285/5626)
AppLovin	95.69%(4949/5172)
Twitter MoPub	55.25%(2227/4031)
Google CrashLytics	98.81%(2158/2184)
Google Analytics	80.77%(1684/2085)
Others	65.81%(28933/43961)
TOT	58.72%(267016/454766)

systematically deny almost all the anonymized events sent by HideDroid. Such a limitation is due to the usage of a proprietary format called *protobuf* [45] to serialize the network requests delivered to the back-end services. Unfortunately, without an a-priory knowledge of the structure of the protobuf request, it is not possible to successfully parse the data [46]. Thus, HideDroid is not able to correctly process the requests, causing a significant drop in the acceptance rate. Still, it is worth noticing that the dynamic testing phase confirms that the failure in the acceptance of the anonymized events does not interfere with the normal execution of any of the AUTs.

Listing 2. Example of an Analytics Request Intercepted by HideDroid

```
{
    "hardware_id":"033ae95da0085566",
    "brand":"Google",
    "device_id":"ffffffff-b626-4582-a9f2-20d36d7a4fe6",
    "model": "Android SDK built for x86",
    "screen_dpi": 560,
    "screen_height": 2701,
    "network": "MOBILE",
    "operator": "T-Mobile",
    "screen_width": 1440,
    "os": "Android",
    "country": "US",
    "language": "en",
    "local_ip": "10.0.2.15",
    "bssid": "02:00:00:00:00:00",
    "ssid": "You are WiFizoned",
    "mac_address": "00:10:FA:6E:38:4A",
    "latest_install_time": 1609930857411,
    "latest_update_time": 1609930857411,
    "first_install_time": 1609930857411,
    "google_advertising_id": "8e83d747-13ec-491f-89bb
-761e9d0cef11",
    "sdk": "android3.0.4",
    "branch_key": "
key_live_pfv4qQtKHbRzXlt5hHufpbmgEBlgiG57",
    "event_type": "AddToCart",
    "data": {
        "contents":"ddr4 memory",
        "id_content":"34",
        "content_type":"pc hardware",
        "price":"300",
    }
}
```

Evaluation of the Anonymization Process. During the experimental phase, HideDroid was able to anonymize more than $200k$ requests. Listing 2 depicts an example of a request generated by an analytics library. The request contains several information regarding the network connection, device, and location. This information can be divided into QID (i.e., *network mode*, *operator*, *country*, *language*, *device*, *model*, *os*, *local IP*, *bssid*, and *ssid*) or EI (i.e., *mac address*, *hardware_id*, and *device_id*). Moreover, the request contains also information about the event generated by the user (i.e., `AddToCart`) and additional details about it (i.e., *contents*, *id_content*, *price*, and *content_type*).

Listing 3 represents the same network request anonymized by HideDroid and exploiting the anonymization pipeline described in Algorithm 1. In detail, the tool relied on the DGH rules to anonymize the information regarding, e.g., brand, device model, network operator, and OS version. All the other information about the user and the device (e.g., `hardware_id`, `device_id`, `local IP`, `bssid`, and `ssid`) are generalized using the generalization procedure described in [1]. Finally, HideDroid replaced the recorded event (i.e., `AddToCart`), with another one taken from the pool of events (i.e., `OpenApp`).

Listing 3. Example of an Analytics Request Anonymized by HideDroid

```
{
    "hardware_id": "033ae9**********",
    "brand": "Smartphone",
    "device_id":"fffffffff-b62
************************",
    "model": "Android",
    "screen_dpi": 500,
    "screen_height": 2700,
    "network": "Mobile Operator",
    "operator": "Anonymous Operator",
    "screen_width": 1400,
    "os": "Smartphone OS",
    "country": "America",
    "language": "en",
    "local_ip": "10.0.0.0",
    "bssid": "02:00:***********",
    "ssid": "You ar***********",
    "mac_address": "00:10:***********",
    "latest_install_time": 1609900000000,
    "latest_update_time": 1609900000000,
    "first_install_time": 1609900000000,
    "google_advertising_id": "8e83d747-13e
************************",
    "sdk": "sdk version",
    "branch_key": "key_live_pfv4q
*************************",
    "event_type": "OpenApp"
}
```

To evaluate the anonymization using local DP techniques, we further inspected the 150 apps that generated most of the analytics events during the dynamic analysis phase. In detail, we replicated the dynamic analysis by stimulating each AUT and recording the anonymized network requests produced by HideDroid in 10 minutes for each of the available privacy levels.

Table 7 reports the results of the analysis on the set of 150 apps. In detail, we reported, for each privacy level, the $Threshold_{action}$ (i.e., TH), the mean number of injected, removed and replaced events (i.e., $\overline{\#Inj_{Ev}}$, $\overline{\#Rem_{Ev}}$ and $\overline{\#Rep_{Ev}}$, respectively), and the mean number of total events (i.e., $\overline{\#Tot_{Ev}}$).

Finally, we computed the KL_Divergence metric [47] (D_{KL}) to evaluate the anonymization process in terms of privacy and utility. This metric allows measuring the *distance* between two distributions of events. A high value of D_{KL}

TABLE 7
Experimental Results of LDP Techniques Used by HideDroid on the 150 Apps Dataset

Privacy	TH	$\overline{\#Inj_{Ev}}$	$\overline{\#Rem_{Ev}}$	$\overline{\#Rep_{Ev}}$	$\overline{\#Tot_{Ev}}$	$\overline{D_{KL}}$
LOW	0.75	8.53	6.44	8.68	37.18	0.05
MEDIUM	0.5	17.64	8.98	17.29	43.76	0.11
HIGH	0.25	26.74	6.09	26.07	55.74	0.19

suggests that the two distributions are very different, i.e., the anonymization process overturns the original data at the expense of its utility. On the other hand, a value equals to 0 indicates that the two distributions are identical, i.e., the anonymization process does not modify the collected data, hence preserving the maximum level of utility.

The last column in Table 7 reports the mean KL_Divergence (i.e., $\overline{D_{KL}}$) between the original distribution of analytics events and each anonymized distribution obtained with the `LOW`, `MEDIUM`, and `HIGH` levels, respectively.

It is worth pointing out that the distance between the original distribution and the anonymized ones is always greater than 0, meaning that the local DP anonymization has successfully increased the privacy of the distribution of events. Also, D_{KL} value is always close to 0, which means that the anonymization process did not overturn the collected data, thereby preserving a reasonable level of utility. Furthermore, the higher is the privacy level, the greater is the D_{KL} value, thereby demonstrating that the utility of the exported data actually lowers when the privacy level rises.

5 Related Work

The wide adoption of third-party analytics libraries in mobile apps has recently attracted the attention of the security research community. The work of Chen *et al.* [8] is one of the first studies focused on the privacy issues related to mobile analytics libraries. In detail, the authors demonstrated how an external adversary could extract sensitive information regarding the user and the app by exploiting two mobile analytics services: Google Mobile App Analytics and Flurry. Moreover, Vallina *et al.* [48] identified and mapped the network domains associated with mobile ads and user tracking libraries through an extensive study on popular Android apps. The authors in [49] highlighted the privacy problem related to a misconfiguration of analytic services. In detail, they proposed PAMDroid, a semi-automated approach to investigate whether mobile app analytic services are actually anonymous and how Attributes Setting Methods (ASMs) can be misconfigured by app developers. These ASMs can be misconfigured by developers so that individual user behavior profiles can be disclosed, which might impose greater privacy risks to users. All the abovementioned works focused on the privacy implications on the usage of analytics libraries and determined that analytics services do not apply any anonymization methodology, thereby highlighting how misconfigurations in those services by the app developers may lead to severe privacy breaches. Their work acted as a motivation for our empirical study to investigate and classify the data collected by analytics service and pushed the design of MobHide and HideDroid. Also, the core of our work is to propose a sound methodology to enhance the privacy of the collected data. However, the identification of application-level or service-level privacy misconfigurations is out of the scope of this work and can be demanded to further extensions. Most of the research activity focuses on proposing novel approaches to enhance user privacy. Beresford *et al.* [50] proposed a modified version of the Android OS called MockDroid, which allows to "mock" the access of mobile apps to system resources. MockDroid allows users to revoke access to specific resources at run-time, encouraging the same users to take into consideration a trade-off between functionality and personal information disclosure. Zhang *et al.* [28] proposed PrivAid, a methodology to apply differential privacy anonymization to the user events collected by mobile apps. The tool replaced the original analytics API with a custom implementation that collects the generated event and applies DP techniques. The anonymization strategy is configured directly by the app developer, which is able to reconstruct at least a good approximation of the distribution of the original events. The authors in [12] proposed an Android app called Lumen Privacy Monitor that analyzes network traffic on mobile devices. This app aims to alert the user if an app collects and sends personally identifiable information (e.g., IMEI, MAC, Phone Number). The application allows the user to block requests to a specific endpoint. To do that, Lumen Privacy Monitor asks for all the Android permissions in order to collect the user data and perform the lookup in the network requests. Zhang *et al.* [51] and Latif *et al.* [52] evaluated the feasibility of the Differential Privacy (DP) approach in the anonymization process of dynamically-created content that is retrieved from a content server and is displayed to the app user. They described how DP could be introduced in screen event frequency analysis for mobile apps, and demonstrated an instance of this approach for Android apps and the Google Analytics framework. Then, they developed an automated solution for analysis, code rewriting, and run-time processing in order to modify the original distribution of screen events preserving, however, the accuracy of the data. Unfortunately, the above solutions do not provide proper data anonymization, thereby proposing either block-or-allow strategies or approaches that enable the reconstruction of the original data by a third-party (e.g., the app developer). Also, most of them require invasive modifications of the apps or the OS (e.g., custom OS and root permissions), and can very hardly be adopted in the wild. To the best of our knowledge, this work is the first proposal that analyzes the usage of analytics libraries in the wild evaluating the real user privacy threats. Moreover, in this work, we also extended our user-centric methodology (proposed in [1]) and described our prototype for Android devices.

6 Conclusion

In this paper, we have analyzed the widespread of analytics libraries and their impact on the privacy of the user and the device by conducting a systematic and automated analysis on the top 4500 Android applications extracted by the Google Play Store.

The obtained results drove us to propose i) an extension of our per-app anonymization methodology - MobHide - and ii) a prototype implementation for the Android ecosystem -

HideDroid - to cope with state-of-the-art mobile analytics frameworks.

The results obtained by our experiments demonstrated that a user-centric solution for anonymizing data collected by analytics libraries is applicable in the wild with a negligible impact on the user and the device.

Still, we advocate that the current methodology can be extended by adopting more sophisticated techniques to adapt to the nature of the identified data. In this respect, we plan to investigate new anonymization strategies such as CAHD [53], l-diversity [54] or t-closeness [55].

Also, we were able to identify several limitations in the adoption of the Android VPN APIs that are, thus, inherited by HideDroid. In particular, if an analytic library enforces SSL Pinning techniques to protect its network traffic, HideDroid is not able to intercept the network requests because the Android app raises an exception due to the invalid certificate. Despite the existence of SSL bypass techniques such as the use of Frida [56], or Xposed [57], they either require root permissions or per-app instrumentation, which may lead to the crash of the AUT. Moreover, if the app developer applies an additional encryption layer on the network traffic, HideDroid will not be able to decrypt the data programmatically. We mitigated such issues in HideDroid by considering encrypted data as generic strings even though the corresponding anonymization process (e.g., the data generalization) would break the decryption process at the backend side. The rationale of such a choice is to prioritize the privacy of the collected information with respect to the utility. To overcome the limitation of such technologies, we plan to investigate the use of virtual environment technologies, such as VirtualApp [58] and DroidPlugin [59], that enable the dynamic hooking of all the events generated by analytic libraries without the need to modify and repack the application. Moreover, by using a virtualization-based approach, we could investigate the extension of MobHide and HideDroid to all the data collected by apps that could affect the privacy of the user.

REFERENCES

[1] D. Caputo, L. Verderame, and A. Merlo, "MobHide: App-level runtime data anonymization on mobile," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur. Workshops*, 2020, pp. 490–507.

[2] Statista, "Number of apps available in leading app stores as of 3rd quarter 2020," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[3] A. Brain, "Android analytics libraries," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://www.appbrain.com/stats/libraries/tag/analytics/android-analytics-libraries

[4] A. Radar, "Improving app ratings and replying to user reviews," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://appradar.com/academy/app-reviews-and-ratings

[5] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the android ecosystem," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1184–1199, May 2020.

[6] Exodus, "Most frequent trackers - Google play," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://reports.exodus-privacy.eu.org/en/trackers/stats/

[7] Y. He, X. Yang, B. Hu, and W. Wang, "Dynamic privacy leakage analysis of Android third-party libraries," *J. Inf. Secur. Appl.*, vol. 46, pp. 259–270, 2019.

[8] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, "Information leakage through mobile analytics services," in *Proc. 15th Workshop Mobile Comput. Syst. Appl.*, 2014, pp. 1–6.

[9] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in Android ad libraries," *Proc. Workshop Mobile Secur. Technol.*, vol. 10, pp. 195–197, 2012.

[10] L. Verderame, D. Caputo, A. Romdhana, and A. Merlo, "On the (un)reliability of privacy policies in Android apps," in *Proc. Int. Joint Conf. Neural Netw.*, 2020, pp. 1–9.

[11] H. Zhang, Y. Hao, S. Latif, R. Bassily, and A. Rountev, "A study of event frequency profiling with differential privacy," in *Proc. 29th Int. Conf. Compiler Construction*, 2020, pp. 51–62.

[12] A. Razaghpanah *et al.*, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018.

[13] G. Navarro-Arribas and V. Torra, "Information fusion in data privacy: A survey," *Inf. Fusion*, vol. 13, no. 4, pp. 235–244, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253512000024

[14] Hidedroid, 2021. [Online]. Available: https://github.com/Mobile-IoT-Security-Lab/HideDroid

[15] D. Caputo, F. Pagano, L. Verderame, and A. Merlo, "Android analytics requests network traffic," 2021. Accessed: Feb. 11, 2021. [Online]. Available: https://www.kaggle.com/x3no21/android-analytics-requests-network-traffic

[16] Facebook, "Facebook analytics," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://developers.facebook.com/docs/app-events/getting-started-app-events-android

[17] Firebase, "Firebase analytics," 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://firebase.google.com/docs/analytics/get-started,?platform=android

[18] Amplitude, "Amplitude analytics," 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://developers.amplitude.com/docs/android

[19] S. D. C. Di Vimercati, S. Foresti, G. Livraga, and P. Samarati, "Anonymization of statistical data," *Inf. Technol.*, vol. 53, no. 1, pp. 18–25, 2011.

[20] T. Li and N. Li, "On the tradeoff between privacy and utility in data publishing," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 517–526.

[21] D. Kifer and J. Gehrke, "Injecting utility into anonymized datasets," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 217–228.

[22] K. Mivule, "Utilizing noise addition for data privacy, an overview," 2013, *arXiv:1309.3958*.

[23] G. Loukides and A. Gkoulalas-Divanis, "Utility-preserving transaction data anonymization with low information loss," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 9764–9777, 2012.

[24] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 6, pp. 1010–1027, Nov./Dec. 2001.

[25] C. Dwork, "Differential privacy: A survey of results," in *Proc. Int. Conf. Theory Appl. Models Comput.*, 2008, pp. 1–19.

[26] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang, "Privacy at scale: Local differential privacy in practice," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 1655–1658.

[27] M. Yang, L. Lyu, J. Zhao, T. Zhu, and K.-Y. Lam, "Local differential privacy and its applications: A comprehensive survey," 2020, *arXiv:2008.03686*.

[28] H. Zhang, S. Latif, R. Bassily, and A. Rountev, "PRIVAID: Differentially-private event frequency analysis for Google analytics in Android apps," Ohio State Univ., Columbus, Ohio, USA, 2018.

[29] Á. Feal, P. Calciati, N. Vallina-Rodriguez, C. Troncoso, and A. Gorla, "Angel or devil,? A privacy study of mobile parental control apps," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 2, pp. 314–335, 2020.

[30] C. Han *et al.*, "The price is (not) right: Comparing privacy in free and paid apps," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 3, pp. 222–242, 2020.

[31] J. Gamba, M. Rashed, A. Razaghpanah, J. Tapiador, and N. Vallina-Rodriguez, "An analysis of pre-installed Android software," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1039–1055.

[32] Androguard, 2019. Accessed: Feb. 11, 2022. [Online]. Available: https://github.com/androguard/androguard

[33] Exodus, "Exodus core library," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://github.com/Exodus-Privacy/exodus-core

[34] Y. Li, Z. Yang, Y. Guo, and X. Chen, "DroidBot: A lightweight UI-Guided test input generator for Android," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion*, 2017, pp. 23–26.

[35] Mitmproxy, 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://mitmproxy.org/

[36] A. Developer, "Android HTTPS," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://developer.android.com/training/articles/security-config#CertificatePinning
[37] Andrea Possemato and Yanick Fratantonio, "Towards HTTPS everywhere on Android: We are not there yet," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 343–360. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/possemato
[38] Nowsecure, "Frida for Android," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://frida.re/docs/android/
[39] C. Wiki, "How can i trust cacert's root certificate?," 2021. Accessed: Feb. 11, 2022. [Online]. Available: http://wiki.cacert.org/FAQ/ImportRootCert
[40] Google, "Android 11 tightens restrictions on ca certificates," 2021. Accessed: Feb. 11, 2022. [Online]. Available: https://httptoolkit.tech/blog/android-11-trust-ca-certificates/
[41] Android blog, "Changes to trusted certificate authorities in Android nougat," 2016. [Online]. Available: https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html
[42] Android documentation, "Network security configuration," 2021. [Online]. Available: https://developer.android.com/training/articles/security-config
[43] Andorid documentation, "Vpnservice," 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://developer.android.com/reference/android/net/VpnService
[44] Google, "Privacy changes in Android 10," 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://developer.android.com/about/versions/10/privacy/changes#proc-net-filesystem
[45] G. Developers, "Protocol buffers," 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://developers.google.com/protocol-buffers
[46] Google, "Language guide (proto3)," 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://developers.google.com/protocol-buffers/docs/proto3
[47] S. Kullback, *Information Theory and Statistics*. Chelmsford, MA, USA: Courier Corporation, 1997.
[48] N. Vallina-Rodriguez *et al.*, "Tracking the trackers: Towards understanding the mobile advertising and tracking ecosystem," 2016, *arXiv:1609.07190*.
[49] X. Zhang, X. Wang, R. Slavin, T. Breaux, and J. Niu, "How does misconfiguration of analytic services compromise mobile privacy?," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1572–1583.
[50] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "MockDroid: Trading privacy for application functionality on smartphones," in *Proc. 12th Workshop Mobile Comput. Syst. Appl.*, 2011, pp. 49–54.
[51] Z. Hailong, L. Sufian, B. Raef, and R. Atanas, "Introducing privacy in screen event frequency analysis for Android apps," in *Proc. 19th Int. Work. Conf. Source Code Anal. Manipulation*, 2019, pp. 268–279.
[52] S. Latif, Y. Hao, H. Zhang, R. Bassily, and A. Rountev, "Introducing differential privacy mechanisms for mobile app analytics of dynamic content," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2020, pp. 267–277.
[53] G. Ghinita, Y. Tao, and P. Kalnis, "On the anonymization of sparse high-dimensional data," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 715–724.
[54] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," *ACM Trans. Knowl. Discov. Data*, vol. 1, pp. 3–es, 2007.
[55] N. Li, T. Li, and S. Venkatasubramanian, "t-Closeness: Privacy beyond k-anonymity and L-diversity," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 106–115.
[56] InfoSec, "The universal SSL pinning bypass for android applications," 2019. Accessed: Feb. 11, 2022. [Online]. Available: https://infosecwriteups.com/hail-frida-the-universal-ssl-pinning-bypass-for-android-e9e1d733d29
[57] Sslunpinning_xposed, 2016. Accessed: Feb. 11, 2022. [Online]. Available: https://github.com/ac-pm/SSLUnpinning_Xposed
[58] Virtualapp, 2022. Accessed: Feb. 11, 2022. [Online]. Available: https://github.com/asLody/VirtualApp
[59] Droidplugin, 2019. Accessed: Feb. 11, 2022. [Online]. Available: https://github.com/DroidPluginTeam/DroidPlugin

Davide Caputo received the PhD degree in computer science from the University of Genova, Italy, in 2022, under the supervision of Alessio Merlo and Luca Verderame. He is currently a cybersecurity engineer with Talos s.r.l.s. His research interests include mobile security and IoT security.

Francesco Pagano received the BSc and MSc degrees in computer engineering from the University of Genova, where he is currently working toward the PhD degree with Computer Security Laboratory, under the supervision of Alessio Merlo and Luca Verderame. His research interests include mobile security and IoT security.

Giovanni Bottino received the BSc and MSc degrees in computer engineering from the University of Genova. Together with his colleague Francesco Pagano and under the supervision of Prof. Alessio Merlo and Davide Caputo, he focused his master's thesis on the issues of Mobile Privacy with the development of the HideDroid app. He is currently a software developer with RINA S.p.A. Company, Genova.

Luca Verderame received the PhD degree in electronic information robotics and telecommunication engineering in 2016, where he worked on mobile security. He is currently an assistant professor of computer engineering with the University of Genoa, Italy. He is also the CEO and cofounder of Talos s.r.l.s., a cybersecurity SME and university spin-off. His research focuses on the security of application ecosystems.

Alessio Merlo is currently an associate professor of computer engineering with the University of Genova, where he leads the Mobile Security Research Group. He has authored or coauthored more than 100 scientific papers in international conferences and journals. His main research interests include mobile and IoT security.