MobHide: App-Level Runtime Data Anonymization on Mobile

Davide Caputo, Luca Verderame, and Alessio Merlo(✉)

DIBRIS - University of Genova, Via Dodecaneso, 35, 16146 Genova, Italy
{davide.caputo,luca.verderame,alessio}@dibris.unige.it

Abstract. Developers of mobile apps gather a lot of user's personal information at runtime by exploiting third-party analytics libraries, without keeping the owner (i.e., the user) of such information in the loop. We argue that this is somehow paradoxical. To overcome this limitation, in this paper, we discuss a methodology (i.e., MobHide), allowing the user to choose a different privacy level for each app installed on her device. According to the user's preferences, MobHide anonymizes the data collected by the analytics libraries before sending them to the app developers, through a fruitful combination of data anonymization techniques. More in detail, the methodology enables to i) analyze all the network traffic generated by the invocation of analytics libraries, ii) anonymize the personal and device data using a *generalization technique*, and the events related to the user's behavior by exploiting *local differential privacy*, and iii) send the anonymized data to the developers.

We empirically assessed the viability of the approach on Android, by implementing the methodology as an Android app, i.e., HideDroid, that relies on the VPN service provided by Google to intercept all network requests. Our preliminary experiments - carried out on a real app (i.e., Duolingo) - are promising, and suggest that runtime data anonymization on mobile is feasible nowadays, as it negligibly impacts the app performance.

Keywords: Android privacy · Analytics libraries · Data anonymization

1 Introduction

In mid-2020 the number of available mobile applications (hereafter, apps) is growing towards 4.5 millions[1] (i.e., 2.56 M Android apps and 1.86 M iOS apps). This fact suggests that the competition among app developers to rise to (or stay

[1] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

This work was partially funded by the Horizon 2020 project "Strategic Programs for Advanced Research and Technology in Europe" (SPARTA).

on) top is always more fierce, as they need to keep building apps that fully meet the user's expectation. To this aim, app developers need to receive continuous feedback on the way users interact with their apps. To achieve such result, they actually keep monitoring both the user's activities and the status of the device in order to *i)* track errors and crashes in the app, *ii)* understand the tastes of the user, and *iii)* deliver personalized advertisements, products or functionalities, in order to maximize the user's experience.

Such monitoring activity is currently carried out at runtime by exploiting *third-party analytics libraries* that enable the collection of information regarding the user's behavior. In detail, such libraries are made of a set of API that allows collecting the user-generated events (e.g., the set of the most visited pages or the history of purchases), and several details about the user herself and the device (e.g., the IMEI number, the OS version, and the GPS location). Developers can include such libraries in the app and invoke their API methods in the app code to log a meaningful event or information. Currently, the most widespread analytics libraries [3,23] are Facebook Analytics[2] and Google Firebase Analytics[3].

However, the adoption of analytics libraries raised serious concerns regarding the user's privacy [14,17] for several reasons. First, as analytics libraries are embedded in the app, they share the app privileges and get access to its resources. Furthermore, analytics libraries do not enforce any privacy-preserving mechanism, as discussed in [11,17,20]. Finally, the user has no control over them: although she can grant or deny the permission to collect personal data, she cannot choose the data to track nor apply any anonymization techniques to her data collected by the analytics libraries. Paradoxically, this means that the management of some user's personal information is devoted to the app developers rather than the user, which is the legal owner. This "status quo" currently maximizes the utility of data (for app developers) at the expense of the user's privacy. To this aim, we argue that the user must be kept in the loop and be free to choose the trade-off between utility and privacy of her own data, before they are delivered to any third-party.

Currently, this problem is gaining momentum, as researchers recently proposed some solutions to try mitigating the privacy issues of third-party libraries at large, and to anonymize the collected personal data. For instance, Zhang et al. [24] proposed a solution allowing the developer to anonymize the collected information according to differential privacy techniques. However, the approach is still developer-centric, i.e., the developer chooses both the anonymization strategy and its configuration. Liu et al. [17] designed an Android app able to intercept and block all the API related to analytics libraries, while Razaghpanah et al. [19] developed an app able to block the network requests that contain personal information. However, both solutions follow an "all or nothing" approach: all personal data are exported in their original form (i.e., maximizing the utility of data), or none of them is exported at all (i.e., maximizing the user's privacy). As data anonymization can be modeled as an optimization problem, where the

[2] https://developers.facebook.com/docs/graph-api/reference/application/activities/.
[3] https://firebase.google.com/docs/analytics/get-started.

aim is to find the optimal balance between data privacy and utility, previous approaches need to be extended further. As a last remark, it is also worth pointing out that the implementation of all the proposed solutions is strongly invasive (i.e., it requires either the adoption of a customized OS, the mandatory presence of root permissions, or the modification of the app logic), and could hardly be adopted in the wild.

Contributions of the Paper. This paper presents a novel, user-centric methodology, called **MobHide**, that allows the per-app anonymization of collected personal data according to a privacy level chosen by the user. In a nutshell, the idea is to collect all the network traffic generated by the invocation of API calls belonging to analytics libraries, and extract the exported data. Then, the next step is anonymizing the personal and device data using a generalization technique, and the data related to the user's behavior using an approach based on the concept of *local differential privacy*, in a way that preserves as much data semantics as possible. Finally, the anonymized data are sent to the expected recipients by mimicking the original network calls.

To prove the effectiveness and the feasibility of MobHide, we implemented the methodology in an Android app called **HideDroid**, and we used it to anonymize the data collected by a real Android app with more than 100M downloads (i.e., Duolingo). HideDroid relies on standard Android APIs to build a VPN-Client that successfully intercepts the network traffic generated by the app with a minimal configuration (i.e., by installing the app certificate). Furthermore, we integrated a transparent repackaging mechanism for the installed apps that do not alter the app behavior, to overcome the network restrictions imposed by the most recent Android OS versions.

Structure of the Paper. The rest of the paper is organized as follows: Sect. 2 introduces the functionalities of analytics libraries, and some basic concepts on data anonymization, while Sect. 3 defines the MobHide methodology. Section 4 presents the HideDroid prototype implementation on Android. Section 5 shows and discusses the usage of our approach on a real app. Section 6 presents the current state of the art, Sect. 7 discusses the limitation of our proposal, while Sect. 8 concludes the paper and points out some extensions of this work.

2 Background

2.1 Notes on Analytics Libraries

Analytics libraries allow to log user's events and device properties during the app execution. There exist several providers of mobile analytics libraries [3]. Among them, Firebase Analytics, Facebook Analytics, and Flurry are largely the most adopted ones [23].

Analytics libraries are composed by two parts, namely i) a Software Developer Kit (SDK) that can be included by developers in the app, and ii) a backend system - usually located in the Cloud - that allows the same developers to track and analyze the collected data through proper control dashboards. The SDK allows the developer to log and monitor either a pre-defined set of standard events or define properly customized events. In general, standard events are common to all apps, and are automatically collected by the SDK and sent to the analytics backend without any further configuration. Examples of such events are "app installation, "app open", and "app close". A custom event is defined by the developer to track app-specific activities. The event is typically represented in a key-value format (e.g., JSON) and sent to the backend by invoking a proper SDK API - typically named `logEvent`. Also, the event often contains some metadata [4].

2.2 Data Anonymization

Data Anonymization (DA) is the process of protecting private or sensitive information by erasing or encrypting identifiers that explicitly connect an individual to some data. For instance, such a process is of paramount importance when companies share data about their users with third parties for analytics or marketing analysis [12]. State of the art DA techniques can be divided into *perturbative* and *non-perturbative*, depending on the kind of data to protect. One of the most widespread *non-perturbative* technique, especially for the multidimensional data (e.g., relational databases), is generalization [21].

Generalization. A piece of information describing an entity (e.g., a user) can be represented by a set of attributes that give details about its features (e.g., gender, date of birth, address). In the original data, where each value is as much specific as possible, each attribute is considered to be in the most specific domain. Generalization techniques consist of replacing the specific value of a set of attributes with a more general one, preserving as much data semantics as possible.

In detail, given an attribute A of a table T, we can define a **domain generalization hierarchy** (DGH) for A as a set of n functions $f_h : h = 0, ..., n-1$ such that:

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} ... \xrightarrow{f_{n-1}} A_n \tag{1}$$

For example, Fig. 1 depicts a set $Z0$ of actual ZIP codes. In such a case, we can define a generalization function f_0 that strips the first rightmost digit to represent a larger geographical area. To make $Z1$ less informative, we can iterate the process and define f_1 and f_2 to strip other digits from the ZIP codes until the most general domain Zn is reached, i.e., where all zip codes are mapped to a singleton value. It is trivial to notice that the more generalization functions are invoked on the original data, the higher is the obtained privacy (and the lower is the data utility), as heterogeneous data are transformed into an always more reduced set of general values.

Generalization techniques are suitable only for semantically independent multidimensional data (e.g., the tuple of a relational database table), but they do not work properly to anonymize sequences of semantically related data. Therefore, they can be used to anonymize the attributes of a single event logged by

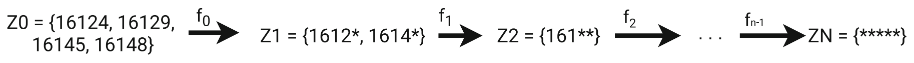


Fig. 1. A sample domain generalization hierarchy (DGH) for ZIP values.

analytics libraries only. To anonymize a sequences of logged (and semantically related) events, we leverage Differential Privacy [13] techniques.

Differential Privacy. In a nutshell, Differential Privacy (DP) applies a perturbation function to a set of related data, e.g., a sequence of events, by using a random noise to alter the original distribution according to a ratio parameter, defined 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 (e.g., an analytics company) that applies DP algorithms and then shares the anonymized dataset with an untrusted third-party client. On the contrary, the *local model* assumes all external entities and communication channels as untrusted. 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 advertising company nor the developer. To this aim, the local DP model is suitable to anonymize sequences of events logged by analytics libraries.

In a *local model*, we can define a sequence of n events such as $e_1, e_2, ..., e_n$ where e_i defines the $i-th$ event. We can assume that all possible values of these events belong to E. A local DP solution can be defined as a perturbation function R that takes as input a sequence of events (i.e., e_i) and outputs another sequence of events (i.e., z_i) different from the previous one. For example, a perturbation function can be a function that adds some noise to the data or replace some events according to a probability defined a priori. The resulting data, i.e., $z_i = R(e_i)$, can be sent to the destination server (e.g., the analytics server). The interested reader can find more details on local DP techniques in [13].

3 The MobHide methodology

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 actual exported data. In principle, we could leverage static analysis techniques, by following, e.g., the techniques we applied in [8], to locate and instrument the methods that invoke analytics libraries APIs. Nonetheless, instrumentation leads to high customization of the app code, requires the systematic repackaging of any app, as well as to deal with potentially obfuscated code [7]. Therefore, MobHide leverages runtime monitoring of any app according to the following steps: i) intercept all data exported by the app through the invocation of API calls belonging to analytics libraries, ii) anonymize data therein by applying the generalization and local DP techniques previously discussed, and iii) send the anonymized data to the backend by mimicking the original network calls. Figure 2 provides a high-level view of the workflow.

In detail, the first step is carried out by the *Privacy Detector* module, which intercepts and filters the traffic that comes from the apps. For each network request belonging to an analytics library API, the module stores it in a buffer repository (*Event Buffer*) and drops the original communication (step 2). Otherwise, the connection is transparently forwarded (step 3).

The *Data Anonymizer* module carries out the anonymization procedure. Periodically, this module pulls the data from the *Event Buffer* (step 4) and applies the anonymization strategy according to the selected app privacy level (step 5) and data generalization hierarchies (step 6). Finally, the anonymized data is sent to the *Data Sender* module (step 7) that forwards them to the expected recipients (step 8). The rest of this section details the different modules and the MobHide anonymization strategies.

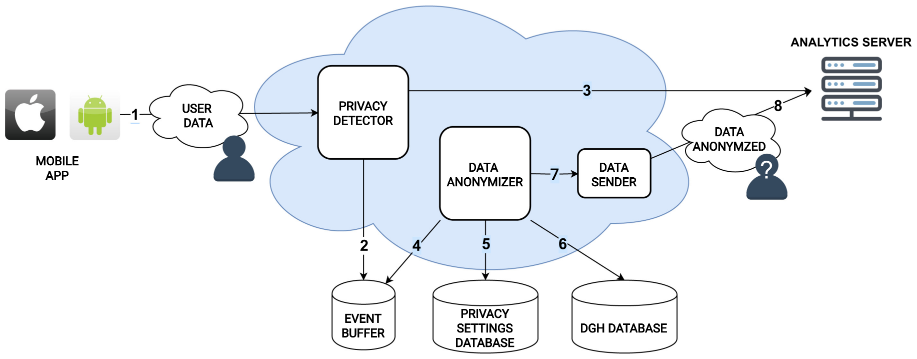


Fig. 2. MobHide - high-level workflow.

3.1 Privacy Detector

The **Privacy Detector** inspects all network traffic coming from the apps selected by the user (Step 1 in Fig. 2). The module parses both encrypted and plain-text traffic according to i) the domain name and ii) the content of the request itself.

In detail, if the domain name belongs to a set of well-known analytic libraries (e.g., `graph.facebook.com` is related to Facebook Analytics, and `app-measurement.com` to Firebase Analytics), the corresponding request is immediately stored in the *Event Buffer*, and the original communication is dropped. If the domain name is not sufficient or unknown, the *Privacy Detector* analyzes the data within the request to identify the parameters and the value most commonly used by analytics libraries. The most common attributes are obtained by the official documentation of the analytics library[4],[5],[6].

Finally, the rest of the network traffic is forwarded to the expected recipients without any further change.

3.2 Privacy Settings Database and DGH Database

MobHide relies on two databases to store the settings defined by the user and the configuration rules for the anonymization strategy.

The privacy level chosen by the user for each app is stored within the *Privacy Settings Database*, and it contains the per-app privacy level defined by the user and thus enables the use of a fine-grained anonymization strategy to each of the apps. The privacy level is mapped into four different values, i.e. `NONE`, `LOW`, `MEDIUM`, `HIGH`. If an app is set to `NONE`, its traffic will be excluded by the anonymization process. On the contrary, the maximum privacy level `HIGH` leads to execute both the generalization and the local DP according to the more restrictive (i.e., privacy-preserving) settings.

The *DGH Database* contains the domain generalization rules for the most common personal attributes collected by the analytics libraries (e.g., gender, date of birth, and location).

3.3 Data Anonymizer

The *Data Anonymizer* is in charge of applying the anonymization strategies on the collected data. As described in Sect. 2.1, the data collected by the analytics libraries includes both the user's in-app actions (i.e., the user's behavior) and information about the user or the device. To deal with such heterogeneous data, the Data Anonymizer builds an anonymization pipeline based on both data generalization and differential privacy techniques.

User's and Device Data Anonymization. To anonymize the information regarding the user and the device, the Data Anonymizer adopts a procedure based on data generalization [21]. The *Data Anonymizer* scans each network request to detect and extract all exporting data. For each attribute, the module looks up for a generalization rule in the *DGH Database*. If a match is found, the value is generalized according to the privacy level. In detail, each increment in the privacy value (i.e., from `LOW` to `HIGH`) implies the application of an extra generalization function of the DGH. In case a match is not found, the Data Anonymizer relies on the following heuristics:

– If the attribute is a **string**, the generalization replaces the last p elements with a generic value $'*'$. The value of p depends on the privacy level, and it is defined as follow:

[4] https://firebase.google.com/docs/analytics/get-started.
[5] https://developers.facebook.com/docs/graph-api/reference/application/activities/.
[6] https://developer.yahoo.com/flurry/docs/.

$$p = \frac{stringLength * selectedPrivacyLevel}{\#PrivacyLevels - 1} \tag{2}$$

where *stringLength* is the string length, *#PrivacyLevels* is the number of available privacy levels (i.e., 4), and *selectedPrivacyLevel* is the privacy level selected by the user (i.e., `NONE`=0, `LOW`=1, `MEDIUM`=2, `HIGH`=3).

– If the attribute in a **number**, the generalization rounds the value to the p most significant digits. The value of p is computed as:

$$p = \frac{\#digits * selectedPrivacyLevel}{\#PrivacyLevels - 1} \tag{3}$$

where *#digits* is the number of digits while the other values are defined in the same way as discussed above.

Anonymization of the User's Behavior. To anonymize the user behavior modeled as a set of related events generated as a consequence of a user action, the Data Anonymizer adopts a heuristic based on local Differential Privacy and the concept of local data perturbation. This heuristic enables the anonymization of the user behavior while preserving structured data for the developer.

The local data perturbation process aims to modify the original behavior distribution by either (i) `removing` intercepted events, (ii) `replacing` events, or (iii) `injecting` crafted events. To do so, the Data Anonymizer relies on a threshold value defined as follows:

$$Threshold_{action} = 1 - \frac{selectedPrivacyLevel}{\#action + 1} \tag{4}$$

where

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

The Data Anonymizer assigns to each intercepted event three pseudo-random numbers (ranging from 0 to 1) that represent the probability of executing one of the three perturbation actions (i.e., inject, remove, replace). Then, the perturbation action is executed only if the corresponding probability is higher than the threshold.

Anonymization Pipeline. The complete procedure for the data anonymization follows the algorithm described in Algorithm 1. For each event stored in the *Event Buffer* (row 3), the algorithm computes the three pseudo-random numbers: $Pr_{inj}, Pr_{rem}, Pr_{rep}$ (rows 4–6).

If the Pr_{inj} is higher than the threshold, the *Data Anonymizer* module builds a new generalized event taken from the pool of the supported event types. If the Pr_{rep} is greater than the threshold (row 11), the module replaces the original event with another valid one. Then, it generalizes the attributes of the replacing event (following the rules described above). Otherwise, the *Data Anonymizer* module checks whether to remove the original event or generalize it. In all three previous cases, the modified event is added to the set of anonymized data (rows 8–9, 12–14, and 17–20), which are returned at the end of the pipeline (row 22).

Algorithm 1. Data Anonymization Pipeline

```
Input: $eventBuffer$, $selectedPrivacyLevel$
Output: $anonymizedEvents$
 1: Initialize $anonymizedEvents \leftarrow$ list()
 2: Initialize $Threshold_{action} \leftarrow 1 - (selectedPrivacyLevel/4)$
 3: for each $event$ in $eventBuffer$ do
 4:     $Pr_{inj} \leftarrow$ rand()
 5:     $Pr_{rem} \leftarrow$ rand()
 6:     $Pr_{rep} \leftarrow$ rand()
 7:     if $Pr_{inj} > Threshold_{action}$ then
 8:         $newGenEvent \leftarrow$ generateNewGenEvent($selectedPrivacyLevel$)
 9:         $anonymizedEvents$.add($newGenEvent$)
10:     end if
11:     if $Pr_{rep} > Threshold_{action}$ then
12:         $replEvent \leftarrow$ replaceEvent($event$)
13:         $replGenEvent.attributes \leftarrow$ generalizeEvent($replEvent.attributes$,
                                           $selectedPrivacyLevel$)
14:         $anonymizedEvents$.add($replGenEvent$)
15:     else if $Pr_{rem} > Threshold_{action}$ then
16:         deleteEvent($event$)
17:     else
18:         $originalGenEvent \leftarrow$generalizeEvent($event.attributes$,
                                     $selectedPrivacyLevel$)
19:         $anonymizedEvents$.add($originalGenEvent$)
20:     end if
21: end for
22: return $anonymizedEvents$
```

3.4 Data Sender

The *Data Sender* module is in charge of forwarding the anonymized data returned by the Data Anonymizer pipeline (step 7) to the analytics backends. To do so, the module mimics the original calls dropped by the Privacy Detector by encapsulating each anonymized data instead of the original plain data (step 8).

4 Implementing MobHide on Android

We empirically assessed the feasibility of MobHide on Android by developing a prototype implementation, called HideDroid, and testing it on a real app.

HideDroid leverages the Android VPN API[7] to capture and analyze the network traffic generated by the apps installed on the device. The app includes a Couchbase Lite[8] NoSql database to implement the *Event Buffer* and two SQLite databases to store the privacy settings and the generalization hierarchies, respectively.

HideDroid Setup. The execution of HideDroid begins by determining the runtime environment, i.e., the OS version and the presence of root permissions. HideDroid implements a transparent SSL/HTTPS proxy [5] to intercept both plain and encrypted network traffic. To this aim, the app generates a self-signed CA and requires the permission to install it in the user's CA store. If the device has root permissions, HideDroid also requires the permission to install the certificate within the system CA store.

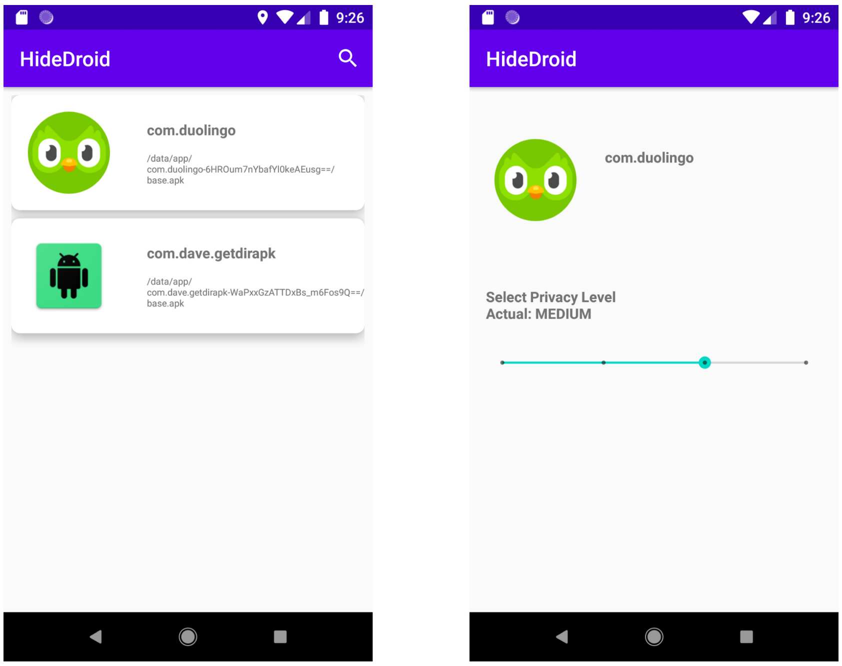
App Privacy Configuration. The HideDroid interface allows the user to view all the apps installed on the device, and select a different privacy level for each app, as shown in Fig. 3. Privacy levels are stored in the *Privacy Setting Database*. For each selected app with privacy level higher than `NONE`, HideDroid checks if the app requires an additional setup to be intercepted by the *Privacy Detector*. It is worth pointing out that this extra step is required only if the Android version is $\geq$7.0, and the user does not have root permissions, due to the current restriction imposed by the OS [1]. Indeed, if the Android OS version is $<$7.0, or if the user accepts the installation of the HideDroid CA in the system CA store, the *Privacy Detector* can intercept the app network traffic without any further customization.

The additional setup step is an *app repackaging* phase, in which proper network configurations are added to the app, without affecting the original app logic. More in details, the repackaging phase is composed of four steps in which HideDroid:

1. unpacks the app using Apktool[9];
2. adds a new network security configuration file[10] to the app, in order to force the usage of the user certificate store (Listing 1.1);
3. modifies the Android manifest file to enable the use of the new network configuration;
4. re-installs the configured app using the `INSTALL_PACKAGES` permission.

At the end of this phase, HideDroid is able to intercept and anonymize the data collected by the analytics libraries.

[7] https://developer.android.com/reference/android/net/VpnService.
[8] https://docs.couchbase.com/couchbase-lite/current/java-android.html.
[9] https://github.com/iBotPeaches/Apktool.
[10] https://developer.android.com/training/articles/security-config.



com.duolingo

/data/app/
com.duolingo-6HROum7nYbafYl0keAEusg==/
base.apk

com.dave.getdirapk

/data/app/
com.dave.getdirapk-WaPxxGzATTDxBs_m6Fos9Q==/
base.apk

com.duolingo

Select Privacy Level
Actual: MEDIUM

(a) Main Interface. (b) Privacy Level Slider.

Fig. 3. Screenshots from the HideDroid prototype.

5 Empirical Assessment

We evaluated the viability of MobHide by executing HideDroid on a real application. After reversing and analyzing a set of most downloaded apps equipped with analytics libraries, we selected Duolingo[11] as a relevant use case for several reasons: first, Duolingo adopts four of the most widespread analytics libraries (i.e., Google Firebase Analytics, Google Crashlitics, Facebook Analytics, and Adjust); furthermore, it requires 30 permissions that can be used to extract information regarding the user and the device (Table 1); finally, it has more than 100M downloads worldwide.

We carried out the experiment on a Huawei P10 device equipped with Android 9.0, an Octa-core (4×2.4 GHz Cortex-A73 & 4×1.8 GHz Cortex-A53), and 4 GB of RAM. Since the experiment involves an Android version $\geq$7.0, the Duolingo app has been repackaged (see Sect. 4). An actual user manually tested the app for two hours, in order to push the invocation of a relevant number of API calls belonging to analytics libraries. During the testing phase, HideDroid captured all network traffic generated by Duolingo, and anonymized the data according to the MobHide strategies described in Sect. 3.

[11] https://play.google.com/store/apps/details?id=com.duolingo&hl=en.

Listing 1.1. The `network_security_config.xml` file injected by HideDroid.

```
<?xml version="1.0" encoding="utf-8"?>
<base-config cleartextTrafficPermitted="true">
    <trust-anchors>
        <certificates src="system" />
        <certificates src="user" />
    </trust-anchors>
</base-config>
```

Table 1. Permissions required by Duolingo.

Permissions	
ACCESS_NETWORK_STATE	BADGE_COUNT_WRITE
AUTHENTICATE_ACCOUNTS	BADGE_COUNT_READ
FOREGROUND_SERVICE	PROVIDER_INSERT_BADGE
GET_ACCOUNTS	BROADCAST_BADGE
INTERNET	WRITE
READ_APP_BADGE	READ
READ_EXTERNAL_STORAGE	WRITE_SETTINGS
RECEIVE_BOOT_COMPLETED	READ_SETTINGS
RECORD_AUDIO	UPDATE_BADGE
VIBRATE	WRITE_SETTINGS
WAKE_LOCK	READ_SETTINGS
WRITE_EXTERNAL_STORAGE	CHANGE_BADGE
UPDATE_COUNT	UPDATE_SHORTCUT
BILLING	READ_SETTINGS
RECEIVE	BIND_GET_INSTALL_REFERRER_SERVICE

We analyzed the network traffic generated by the advertising libraries. Regarding the user's and device profiling, the `model of device`, the `network latency`, the `username`, and the `free space on disk` are the most captured information. Also, the app collected a set of events that describes the user's behavior. Examples of such events include `app_open`, `app_install`, and `learning_reason_tap`. During the two-hours experiment, HideDroid collected 123 events belonging to 39 different classes. Figure 4 summarizes the frequency of each captured event, while Listing 1.2 shows a subset of actual personal data collected by Duolingo and exported to the analytics backend.

We tested all the available privacy levels on Duolingo, in order to evaluate the anonymization capabilities of HideDroid. As described in Sect. 3, MobHide performs two types of anonymization for personal and device information and for user's events, respectively. Listing 1.3 shows an example of the data anonymized after applying a generalization technique with the privacy level set to `HIGH` to the original data showed in Listing 1.2: note that all the string values have been converted to a sequence of * (e.g., "client_id"), while the integer parameters have been rounded to the most meaningful digit (e.g., "memory_maximum").

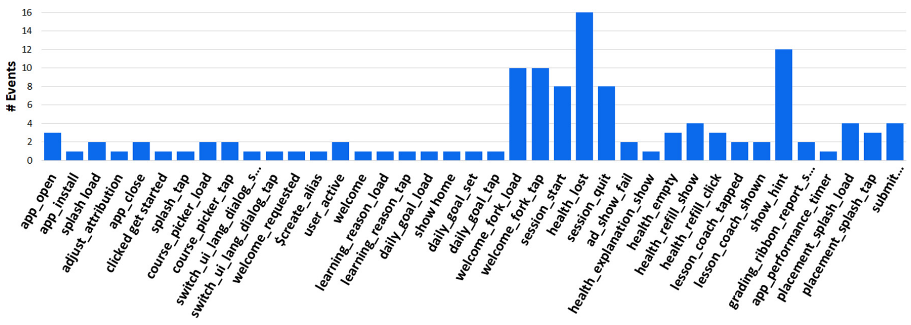


Fig. 4. Distribution of the user's events collected by Duolingo during the two-hours experiment.

```
...
" event_type ": " app_open ",
" event_timestamp": 1591880722000,
" client  ": {
    " client_id  ": " android−excess "
},
" attributes": {
    " memory_maximum ": 268435456,
    " memory_class ": 96,
    " memory_system_available ": 2669375488,
    " data_saver ": " enabled ",
    " memory_class_large ": 256,
    " $screen_height ": 1794,
    " $app_release ": 951,
    " memory_system_total ": 3156844544,
    " screen_width ": 411,
    " $carrier  ": " Android ",
    " Client  ": " Duodroid ",
    " orientation  ": " portrait  ",
    " mp_lib ": " android ",
    ...
```

Listing 1.2. Example of event collected by Duolingo.

```
...
" event_type ": " app_open ",
" event_timestamp ": 1591880722000,
" client  ": {
    " client_id  ": "************"
},
" attributes": {
    " memory_maximum ": 200000000,
    " memory_class ": 90,
    " memory_system_available ": 2000000000,
    " data_saver ": " undefined ",
    " memory_class_large ": 200,
    " $screen_height ": 1000,
    " $app_release ": 900,
    " memory_system_total ": 3000000000,
    " screen_width ": 400,
    " $carrier  ": " undefined ",
    " Client  ": " undefined ",
    " orientation  ": " undefined ",
    " mp_lib ": " undefined ",
    ...
```

Listing 1.3. Example of anonymized event with privacy level `HIGH`.

Figure 5 shows the distributions of the anonymized event frequencies for each levels of privacy (i.e., `NONE`, `LOW`, `MEDIUM`, `HIGH`). It is worth noticing that each privacy level has its own specific distribution pattern. To prove that the distributions are actually different from each other, we computed the KL_Divergence [15] (i.e., D_{KL}) which allows measuring the *distance* between two distributions. A high value of D_{KL} suggests that the two distributions are very different, while $D_{KL} = 0$ indicates that two distributions are identical. We calculated D_{KL} between the original event distribution and each anonymized distribution. The results are reported in Table 2.

Table 2. Parameters and metrics of the HideDroid anonymization phase.

Privacy	TH	$\# \, Inj_{Ev}$	$\# \, Rem_{Ev}$	$\# \, Rep_{Ev}$	$\# \, Tot_{Ev}$	D_{KL}	Ex. Time
LOW	0.75	24	35	28	140	0.11	0.416
MEDIUM	0.5	62	61	66	190	0.28	0.352
HIGH	0.25	94	93	98	223	0.38	0.419

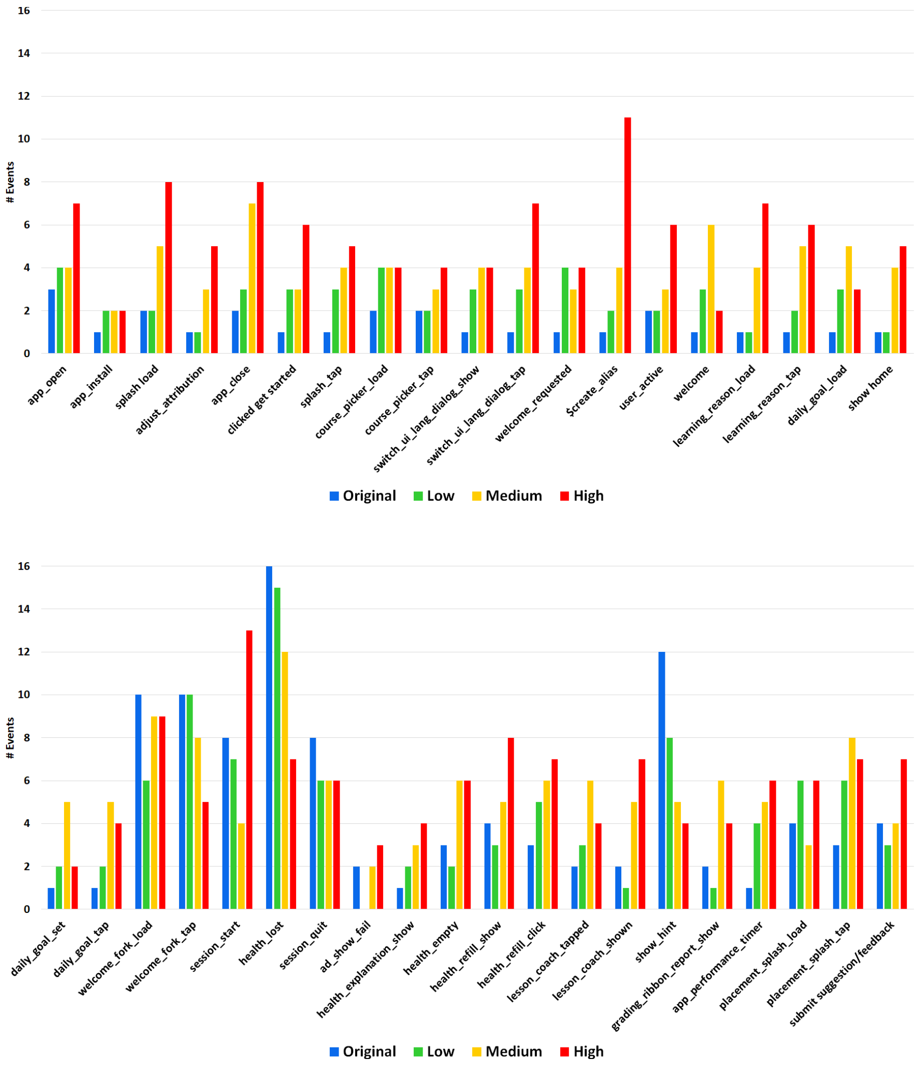


Fig. 5. Comparison of the different event distributions generated by HideDroid, according to the selected privacy level.

Such a table summarizes the results of the anonymization phase on the set of events for each privacy level. In detail, the first column indicates the privacy level (i.e., Privacy), while columns 2 to 6 describe the parameters and the metrics for the local DP, i.e., the $Threshold_{action}$ (i.e., TH), the number of injected, removed and replaced (i.e., # Inj_{Ev}, # Rem_{Ev} and # Rep_{Ev} respectively) events, the number of total events (i.e., # Tot_{Ev}). Column 7 contains the value of the KL_Divergence, while the last columns contain the execution time (i.e., Ex. Time) required to anonymize the list of events.

Regarding D_{KL}, it is worth pointing out that the distance between the original distribution and the anonymized ones is always greater than 0. Furthermore, the higher is the privacy level, the greater is the D_{KL} value, thereby suggesting that the utility of the exported data lowers when the privacy level rises.

Performance (i.e., Ex. Time) is likewise very promising. In fact, it is worth noticing that the anonymization of a data flow belonging to 2 h of app usage and that contains more than 120 events, requires less than a second. Albeit further studies are required, this suggests that the on-the-fly execution of data anonymization techniques at the state of the art on mobile could be feasible on (most of) the current mobile devices.

6 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. [11] is one of the first studies that explicitly focus 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, i.e., Google Mobile App Analytics and Flurry. Moreover, Vallina et al. [22] identified and mapped the network domains associated with mobile ads and user tracking libraries through an extensive study on popular Android apps.

Still, most of the research activity focus on proposing some novel approaches to enhance privacy. For instance, Beresford et al. [10] 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. [25] 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 can reconstruct at least a good approximation of the distribution of the original events.

The authors in [19] 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.

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, MobHide is the first proposal that allows the user to choose a per-app privacy level and, at the same time, granting the possibility to export anonymized data. Furthermore, our prototype HideDroid has been designed to ensure minimal invasiveness on the mobile device.

7 Discussion and Future Developments

This work aims to demonstrate the feasibility of runtime anonymization of personal data exported by mobile apps and the viability of allowing users to choose a level of privacy for each installed app. Nonetheless, both our methodology (i.e., MobHide) and implementation (i.e., HideDroid) have some limitations.

In the current definition, the MobHide methodology adopts basic - yet effective - DA techniques on the collected data. Still, an extensive evaluation of the type of data transmitted by third-party analytics libraries could unveil complex structures (e.g., multidimensional data, time-series, transaction data, ...). To this aim, other - more complex - DA techniques, such as k-anonymity [21], l-diversity [18] or t-closeness [16], must be taken into consideration and implemented in HideDroid.

Moreover, the traffic recognition capabilities of MobHide are based on a predefined mapping between the hosts and the corresponding analytics services. If an app sends data to an unknown host, MobHide tries to recognize whether the request belongs to an analytic service, according to a keyword-based heuristic (e.g., if the word "event" is contained in the network request). However, such a technique could introduce some false positives, leading to potential app malfunctioning if the request contains data related to the logic of the app. Also in this case, an extensive analysis of such heuristic in the wild will allow evaluating its reliability. In case of low reliability, the adoption of ML-based network recognition techniques [22] could be taken into consideration.

Regarding the limitations of the prototype implementation, HideDroid has been designed to minimize the impact on the target apps. Indeed, we developed the tool with the aim to reduce as much as possible the app customization, and, therefore, we rely on app repackaging only on devices equipped with Android *geq* 7.0 and without root permissions. However, the repackaging process may fail against system apps or apps with anti-repackaging mechanisms in place. Also, the presence of certificate-pinning mechanisms applied to the network traffic of analytics libraries could interfere with the ability of HideDroid to analyze and anonymize the corresponding data.

To overcome the above technical limitations, we plan to evaluate the usage of DroidPlugin [2] or VirtualApp [6] virtual environments that provide the ability to intercept the network traffic without the need of any app customization.

8 Conclusion

In this paper, we introduced MobHide, the first "user-centric" methodology for the per-app anonymization of the data collected by third-party analytics libraries. Furthermore, we proposed HideDroid, a prototype implementation for Android that has been tested on a real-world app with more than 100M downloads.

This work is a first step towards balancing between data utility and user privacy in mobile ecosystems, demonstrating the feasibility of introducing data anonymization locally, i.e., directly on the mobile device without the need for an external trusted party.

Albeit promising, the results suggest that an extensive assessment campaign is needed to tune the proposed anonymization pipeline. As a first step in this direction, we intend to include the support of other third-party libraries and generalization heuristics, and to use Trusted Execution Environment (TEE) technologies [9] to protect the confidentiality and integrity of the collected data. Finally, we plan to release HideDroid on the Google Play Store by the end of 2020.

References

1. Android 7.0 news. https://developer.android.com/about/versions/nougat/android-7.0#network_security_config. Accessed 27 May 2020
2. Droidplugin. https://github.com/DroidPluginTeam/DroidPlugin. Accessed 27 May 2020
3. Exodus privacy. https://reports.exodus-privacy.eu.org/en/trackers/stats/. Accessed 27 May 2020
4. Firebase log event. https://firebase.google.com/docs/reference/android/com/google/firebase/analytics/FirebaseAnalytics.Event. Accessed 27 May 2020
5. Transparent proxy TLS. https://docs.mitmproxy.org/stable/concepts-modes/. Accessed 27 May 2020
6. VirtualApp. https://github.com/asLody/VirtualApp. Accessed 27 May 2020
7. Aonzo, S., Georgiu, G.C., Verderame, L., Merlo, A.: Obfuscapk: an open-source black-box obfuscation tool for android apps. SoftwareX **11**, 100403 (2020). https://doi.org/10.1016/j.softx.2020.100403, http://www.sciencedirect.com/science/article/pii/S2352711019302791
8. Armando, A., Costa, G., Merlo, A., Verderame, L.: Enabling BYOD through secure meta-market, pp. 219–230 (2014). https://doi.org/10.1145/2627393.2627410
9. Armando, A., Merlo, A., Verderame, L.: Trusted host-based card emulation. In: 2015 International Conference on High Performance Computing & Simulation (HPCS), pp. 221–228. IEEE (2015)

10. Beresford, A.R., Rice, A., Skehin, N., Sohan, R.: MockDroid: trading privacy for application functionality on smartphones. In: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, HotMobile 2011. Association for Computing Machinery, New York (2011)
11. Chen, T., Ullah, I., Kaafar, M.A., Boreli, R.: Information leakage through mobile analytics services. In: Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (2014)
12. Cormode, G., Srivastava, D.: Anonymized data: generation, models, usage. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (2009)
13. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Found. Trends® Theor. Comput. Sci. **9**(3–4), 211–407 (2014)
14. He, Y., Yang, X., Hu, B., Wang, W.: Dynamic privacy leakage analysis of android third-party libraries. J. Inf. Secur. Appl. **46**, 259–270 (2019)
15. Kullback, S.: Information Theory and Statistics. Courier Corporation, North Chelmsford (1997)
16. Li, N., Li, T., Venkatasubramanian, S.: t-Closeness: privacy beyond k-anonymity and l-diversity. In: 2007 IEEE 23rd International Conference on Data Engineering. IEEE (2007)
17. Liu, X., Liu, J., Zhu, S., Wang, W., Zhang, X.: Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. IEEE Trans. Mob. Comput. **19**(5), 1184–1199 (2020)
18. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data (TKDD) **1**(1), 3 (2007)
19. Razaghpanah, A., et al.: Apps, trackers, privacy, and regulators: a global study of the mobile tracking ecosystem (2018)
20. Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen, H.: Investigating user privacy in android ad libraries
21. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. Int. J. Uncertainty Fuzziness Knowl. Based Syst. **10**(05), 571–588 (2002)
22. Vallina-Rodriguez, N., et al.: Tracking the trackers: towards understanding the mobile advertising and tracking ecosystem. arXiv preprint arXiv:1609.07190 (2016)
23. Verderame, L., Caputo, D., Romdhana, A., Merlo, A.: On the (un)reliability of privacy policies in android apps. In: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2020), Glasgow, UK, July 2020
24. Zhang, H., Hao, Y., Latif, S., Bassily, R., Rountev, A.: A study of event frequency profiling with differential privacy. In: Proceedings of the 29th International Conference on Compiler Construction, CC 2020. Association for Computing Machinery, New York (2020)
25. Zhang, H., Latif, S., Bassily, R., Rountev, A.: Privaid: Differentially-private event frequency analysis for google analytics in android apps