Leave my Apps Alone! A Study on how Android Developers Access Installed Apps on User's Device

Gian Luca Scoccia
gianluca.scoccia@univaq.it
DISIM, University of L'Aquila
L'Aquila, Italy

Ibrahim Kanj
i.kanj@student.vu.nl
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Ivano Malavolta
i.malavolta@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Kaveh Razavi
razavik@ethz.ch
ETH Zürich
Zürich, Switzerland

ABSTRACT

To enable app interoperability, the Android platform exposes *installed application methods* (IAMs), i.e., APIs that allow developers to query for the list of apps installed on a user's device. IAMs require no Android permissions and it is known that information collected through these methods can be used to precisely deduce end-users interests and personal traits, thus raising privacy concerns. In this paper, we present a large-scale empirical study investigating the presence of IAMs in Android apps and their usage by Android developers.

Our results highlight that: (i) IAMs are widely used in commercial applications while their popularity is limited in open-source ones; (ii) IAM calls are mostly performed in included libraries code; (iii) more than one-third of libraries that employ IAMs are advertisement libraries; (iv) a small number of popular advertisement libraries account for over 33% of all usages of IAMs by bundled libraries; (v) developers are not always aware that their apps include IAMs calls.

Based on the collected data, we suggest revising the way IAMs are currently managed by Android, introducing either an ad-hoc permission or an opt-out mechanism, to improve both developer and end-user awareness.

KEYWORDS

Android, Apps, Privacy

ACM Reference Format:
Gian Luca Scoccia, Ibrahim Kanj, Ivano Malavolta, and Kaveh Razavi. 2020. Leave my Apps Alone! A Study on how Android Developers Access Installed Apps on User's Device. In *IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems (MOBILESoft '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3387905.3388594

MOBILESoft '20, October 5–6, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7959-5/20/05…$15.00
https://doi.org/10.1145/3387905.3388594

1 INTRODUCTION

The Android platform provides a wide range of APIs to application developers, to allow for the creation of feature-rich apps that take full advantage of the device and platform capabilities [26]. Among others, to enable app interoperability, APIs are given to allow for retrieving various information related to the applications that are currently installed on the device [13]. From the users' point-of-view these methods are *silent*, as no special authorization is required for their usage and they provide no visual indication during their operation. Therefore, typical users are usually not aware that such methods do exist. Hereafter we will refer to these methods as *Installed Application Methods* (IAMs).

Nowadays, the average smartphone user has over 60 apps installed on her device [2], each chosen on the basis of her own interests and personal traits (e.g., gender, spoken languages, religious beliefs). Given that the list of installed applications is readily available to developers, it is natural to wonder the extent in which the users' traits can be deducted from it. Past research, discussed in Section 2, has shown that many of these traits can be inferred with near-optimal accuracy. Hence, IAMs prompt privacy concerns.

However, to this day, no inquiry has been conducted on the prevalence of IAMs in Android apps and how they are employed by Android developers. In this paper, we fill this gap, investigating **how IAMs are used by Android developers**. We aim to assess the scale of IAMs usage and provides insights on the reasons behind their popularity.

To this end, we conducted a large-scale empirical study on 14,342 free Android apps published in the Google Play Store and 7,886 open-source Android applications. We identify among them applications that employ IAMs and extract from them information such as fields accessed through these APIs and whether the call is performed in the app's own code or by an included library. Furthermore, we manually identify the main purpose of the most popular libraries found to be using IAMs. Additionally, we perform an assessment of developers' knowledge and awareness about the presence of IAMs in their apps by means of an online questionnaire. Finally, building from the collected data, we discuss the open issues connected with IAMs, (e.g., widespread use in advertisement libraries, lack of developer awareness) and we suggest some changes to the Android platform to address them.

The **main contributions** of this study are the following:

- Empirical results about the usage of IAMs, by analyzing their usage in 14,342 free Android apps published in the Google Play Store and 7,886 open-source Android applications;
- An investigation of developers' awareness regarding the presence of IAMs in their apps, conducted by means of an online questionnaire filled in by 70 participants;
- A discussion of the issues emerging from the collected data, including suggested changes to the Android platform and open research directions.

The **target audience** of this paper is composed of privacy-aware users, researchers and Android platform maintainers. We support users by providing them with a set of recommendations to employ during app selection to minimize privacy risks. We support researchers by (i) providing a characterization of how IAMs are used in practice, and (ii) eliciting from collected data existing issues and open research directions. Lastly, we support Android platform maintainers by suggesting some changes to the Android platform aimed at increasing developers' awareness and end-users' control over IAMs, according to collected data.

The remainder of this paper is structured as follows. Section 2 provides background concepts about IAMs and privacy concerns related to their usage. Section 3 describes the design of our study. Section 4 presents the main results of our study, that are are discussed in Section 5. Section 6 discusses the threats to the validity of our study, whereas Section 7 describes related work. Section 8 closes the paper.

2 BACKGROUND

IAMs are provided by the `packagemanager` class, that exposes two methods for retrieving various kinds of information related to the application packages that are currently installed on the device: `getInstalledApplications()` and `getInstalledPackages()` [13]. The difference between the two methods is slight[1]: the former is restricted to provide information declared inside the `Application` tag in the apps' manifest file, while the latter is more general and can return all information declared in the manifest file, such as employed services, declared activities, meta-data, etc. It is important to note that currently these methods are not classified as sensitive APIs in the Android platform [14]. Hence, their usage inside applications is silent to the outside: declaring specific permissions is not required and it is not mandatory to notify end-users.

```
1   // Let's look for a calculator application
2   mCalculatorActivityItems = new ArrayList<HashMap<String,
          Object>>();
3   mPackageManager = getPackageManager();
4   List<PackageInfo> packs = mPackageManager.
          getInstalledPackages(0);
5   for (PackageInfo pi : packs) {
6     if (pi.packageName.toLowerCase().contains("calcul")) {
7       HashMap<String,Object>map = new HashMap<String,Object>();
8       map.put("appName",
9         pi.applicationInfo.loadLabel(mPackageManager));
10      map.put("packageName", pi.packageName);
11      mCalculatorActivityItems.add(map);
12    }
13  }
```

Listing 1: Example usage of getInstalledApplication()

An example usage of these methods is provided in Listing 1, extracted from app `ph.coreproc.android.philippineincometax`. In the listing, after initializing required classes and data structures (lines 1-3), the app retrieves the list of packages installed on the device (line 4) and iterates on it (lines 5-13). During the iteration, the package name of each installed app is compared to a predefined string (line 6) to identify calculator apps installed on the device. The package name (lines 10) and their human-readable counterpart (lines 8-9) of these apps is stored in a list for future use (line 11).

Since installed apps can readily be collected through IAMs, researchers have investigated whether one user's traits can be extrapolated from their installed apps list. Seneviratne et al. [38] have been the first to investigate this question, showing that using a single snapshot of a user's installed apps, their gender can be instantly predicted with an accuracy around 70%, by training a classifier using established supervised learning techniques. In a subsequent development [39], they extend their classification techniques to other traits such as religion, relationship status, spoken languages and countries of interest. Malmi et al. [25] study the predictability of user demographics (e.g., age, race, and income) from installed applications under varying conditions. In their study, gender proved to be the most predictable attribute (82.3% accuracy), whereas income proved to be the hardest (60.3% accuracy). Moreover, training set size and the number of apps on the user device can have an impact of over 10% on the prediction accuracy. Interestingly, in their experiments, the quality of predictions significantly drops for users with more than 150 apps installed. Frey and colleagues have investigated the usage of the information collected from IAMs to predict users' significant life events (e.g., marriage, first car, becoming a parent) [16]. Compared to a random model, their prediction system achieves significantly higher accuracy (up to 87.1%). Hence, they suggest that their findings are potentially useful for companies to identify and target possible customers. Demetriou et al. [11] investigated the extent to which information provided by IAMs can be leveraged by embedded advertising libraries to infer user traits when combined with other information extracted from the host app files and run-time inputs. Their results show that traits such as age, sex, and marital status can be inferred with over 90% precision and recall.

It is important to notice that IAMs are not exclusive to Android. Similar methods also exist in Apple's iOS, currently the second most popular mobile operating system [21]. However, in recent versions of the operating system, applications of interest have to be preemptively declared inside the app own manifest file, and thus are reviewed by app store moderators before publication[2].

3 STUDY DESIGN

This section describes how we designed our study. In order to perform an objective and replicable study we followed the guidelines on empirical software engineering in [51] and [40].

In order to allow independent verification and replication of the performed study, we make publicly available a full replication package containing (i) the Python scripts for data extraction and

[1] https://stackoverflow.com/questions/8720545/getinstalledapplications-vs-getinstalledpackages

[2] https://cromulentlabs.wordpress.com/2016/01/15/explanation-of-canopenurl-changes-in-ios-9/

analysis, the obtained raw data, and the Java files of the apps we used as subjects.[3]

3.1 Goal and research question

The *goal* of this paper is to understand how IAMs are used in practice, for the purpose of gaining insights on what measures can be introduced to improve end-users privacy protection. The *context* of this study includes 14,342 free Android apps published in the Google Play Store and 7,886 open-source Android applications. We refined this goal into the following research questions:

RQ1 – How are usages of IAMs distributed across app categories?

RQ2 – What kinds of information are most frequently accessed through IAMs?

RQ3 – How are usages of IAMs distributed between app code and included libraries code?

RQ4 – What is the declared main role that libraries calling the IAMs play?

RQ5 – To what extent are developers aware of IAMs and tend to reflect that awareness?

RQ1 aims to measure how common is the use of IAMs and, at the same time, highlight differences in their adoption across different app categories. RQ2 intends to appraise what information is commonly accessed through IAMs, in order to gain insights on their practical use. The purpose of RQ3 is to measure how many calls to IAMs are being initiated from applications' own code and how many are being initiated from included libraries code. RQ4 wants to appraise what is the main role played by libraries that performs IAMs calls. RQ5 intents to assess how aware are developers of the sensitiveness of IAMs and, consecutively, how responsible are they in their usage.

3.2 Data collection

Figure 1 provides an overview of our data collection process, as well as of the subsequent procedures performed to extract data relevant to our research questions (explained in detail in Section 3.3).

To answer our first four research questions we relied on two different datasets of, respectively, open-source and commercial Android apps. We chose AndroidTimeMachine [18] as a starting point for the collection of the former. AndroidTimeMachine contains information about 8,431 real-world open-source Android apps, verified to be published on the Google Play Store. It provided us with (i) URLs to apps Git repositories from which we could obtain the full commits history and, (ii) metadata extracted from the Google Play store, such as app category and ratings. From it we were able to collect source code files of 7,886 open-source Android applications (the remainder are no longer publicly available on Github).

As a starting point for the collection of the commercial apps dataset we considered the top 500 most popular free apps from each of the 35 categories of the Google Play Store, according to the App-Annie service for app ranking analysis[4] as of 21 April 2019. A total of 17,164 unique apps were identified, after removing duplicates that appear in multiple categories. Afterward, binary files (i.e., the APKs) for the latest version of each app were collected from Androzoo [1]. Binaries for a total of 14,342 apps were collected this way. Notice that an app might appear in both datasets. However, potentially, the commercial app version can differ from the open-source one, as the developer might include additional (proprietary) code into his project before publication on app stores. Hence, we chose to abstain from the removal of duplicates that appear in both datasets.

To answer RQ5 we also relied on a short developer questionnaire, sent to all the 4,227 app developers that were found to be using IAMs in previously mentioned datasets. Authors' email addresses were extracted from Github commits and apps description pages on the Google Play Store. No compensation was offered in exchange for answering the questionnaire. The structure of the questionnaire is detailed in the following. It is comprised of three questions:

Q_1: *Where* does your app use the getInstalledApplications() or getInstalledPackages() APIs?

Q_2: *Why* does your app use the getInstalledApplications() or getInstalledPackages() APIs?

Q_3: Do you want to add any comments relevant for this study?

We chose to keep the number of questions limited to reduce the time required to complete the questionnaire and, in turn, minimize the number of incomplete answers. Q_1 is a multiple choice question and possible answers to it are: *"In core functionalities of the app"*, *"In a third-party library"*, *"They are not used at all"*, and *"Other"*. It is mandatory to provide an answer and participants are invited to type their own answer if *"Other"* is selected. Q_2 is instead an open question and it is also mandatory. Q_3 is an open question too but answering it is not required. Notice that, since the questionnaire is only forwarded to developers of apps that have been found to use IAMs, a developer answering *"They are not used at all"* to Q_1 reveals his unawareness about the presence of IAMs in the app. For this reason, we always require a mandatory answer to Q_2, as it can provide insights on the reasons behind this lack of awareness when the developer declares that IAMs are not used in the app.

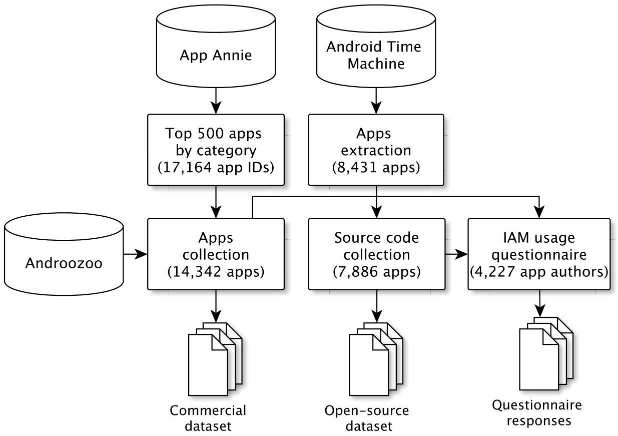
App Annie
Android Time Machine
Top 500 apps by category (17,164 app IDs)
Apps extraction (8,431 apps)
Androozoo
Apps collection (14,342 apps)
Source code collection (7,886 apps)
IAM usage questionnaire (4,227 app authors)
Commercial dataset
Open-source dataset
Questionnaire responses

Figure 1: Data collection and extraction

[3] https://github.com/S2-group/mobilesoft-2020-iam-replication-package
[4] www.appannie.com/apps/google-play/top-chart/united-states

3.3 Data extraction

We extracted relevant data for answering our research questions from our datasets. For this purpose, we identified and recorded occurrences of calls to IAMs from the source code of both open-source and commercial apps. While this process was straightforward for the former, for the latter we relied on decompilation to extract the source code from collected binaries. For this task, we adopted a sequence of two off-the-shelf tools: `dex2jar`[5] and `JD-Core`[6]. The first was used to unpack binaries and extract java class files and the second to decompile class files to java source code. Notice that, although these are state-of-the-art tools, in some cases this process can fail. For this reason, we were unable to decompile 782 files out of the 14,342 commercial applications. Hence, collected data has to be considered as a lower bound of actual IAMs usage for the commercial dataset.

To answer RQ2, we have recorded, for each IAMs call, the fields that were accessed on the returned applications list. For this task we rely on srcML [6] to transform the source code into a traversable `XML` representation. We limit the scope of our field extraction to the file where the IAM call appears as the size of our dataset is unfeasible for the application of whole-app static analysis tools, due to their high processing and memory requirements [3, 49].

Similarly, to answer RQ3 and RQ4, we extracted the *package name* from its declaration at the beginning of the java source code file from which the IAM call is performed. We consider the IAM call to be originating from the app's own code if the extracted package name does contain, as its prefix, the app main package name, declared in the app manifest file (e.g., `apalon.weatherlive.updater` matches with `apalon.weatherlive`). It is considered as originating from a library otherwise.

To answer RQ5 we extrapolate insights from answers to our developers questionnaire. In particular, answers to Q_1 can provide *quantitative* insights while answers to Q_2 and Q_3 can lead to *qualitative* insights.

Limitations of our data extraction approach are discussed in Section 6.

3.4 Analysis

To provide an answer to RQ1 we resort to descriptive statistics, computing counts and usage rates of IAMs across different datasets and different app categories (as defined in the Google Play Store). Likewise, to answer RQ2 and RQ3 we compute similar statistics for accessed fields and usages of IAMs in libraries.

To answer RQ4, we need to assess the declared main role of libraries adopted in our datasets that employ IAMs. As, to the best of our knowledge, there is no existing automated technique able to perform this task, we define a manual procedure to determine the declared main role of an included library and we employ it to analyze a sample of our data. The procedure, for each library to be analyzed, is as follows: (i) input the library package name on a web search engine to trace back its official website (or repository); (ii) manually survey the website to infer the library main role; (iii) synthesize it into an informative label following the guidelines of descriptive coding [34]. The intuition behind the technique is that the declared main role of a library can be inferred relatively quickly and easily from its official website, as most libraries websites are built to concisely and effectively convey their purpose to potential adopters. In cases where searching for the library package name does not lead to the immediate identification of its official website, we recursively repeat the search with progressively smaller package name substrings.

We obtain a sample of our data, reasonably sized for manual analysis, through *purposive sampling* [19], with the ultimate goal in mind of maximizing the coverage of our analysis. To this purpose, we decided to include in our sample all the libraries that were employed by at least five different apps in our datasets. In other words, our sampling rule gives precedence to popular, widely adopted libraries. This led to the identification of 154 individual libraries, that account for 82.83% of all in-library IAMs usages in our dataset (68.64% of all IAMs usages in our datasets).

To reduce bias, two different researchers independently analyzed the complete sample. After completing the analysis, the two aligned the labels with each other, solved all the cases in which there was a disagreement, and grouped similar labels. We measure agreement between the two, before solving disagreement cases, using the Krippendorff's Alpha [24], resulting in an $\alpha = 0.868$. We choose this measure for its ability to adjusts itself to small sample sizes. Values of α above 0.8 are considered as an indication of reliable agreement [23]. The disagreements were mostly due to the fact that one of the two involved researchers adopted more general labels in his initial coding (e.g., *Utility* in place of *Analytics*). After discussing disagreements, the two coders agreed on adopting the more specific labels.

In relation to RQ5, we once more rely on descriptive statistics to analyze answers to Q_1, while instead we resort on manual qualitative content analysis [27] for answers to Q_2 and Q_3 .

4 RESULTS

In this section, we disclose the results of our analysis grouped accordingly to the research questions presented in Section 3.

4.1 RQ1: How are usages of IAMs distributed across app categories?

The plot in Figure 2 provides an overview of IAMs usages in both commercial and open-source apps. Categories for which no apps were collected are marked with the symbol "–". IAMs usages appear to be considerably more common in commercial apps, with a total of 4,214 apps employing them, amounting to 30.29% of the total. Conversely, a total of 228 apps employ IAMs in open-source apps, amounting to only 2.89% of the total. Focusing on commercial apps, we can notice that usages of IAMs occur in all categories. However, distribution of usages varies greatly among categories: over half of analyzed apps employ IAMs in categories *Games* (72.97%), *Comics* (70.50%), *Personalization* (60.6%) and *Auto & Vehicles* (57.61%) while usages diminish to about one in ten apps in categories *Medical* (14.36%), *Libraries & Demo* (12.26%) and *Events* (11.90%). Regarding open-source apps, usages appear to be less frequent, being more common in categories such as *Personalization* (8.95%), *Shopping* (6.67%) and *Tools* (5.88%) while completely absent in other categories such as *Food & Drink*, *Events* and *Comics*. To summarize:

[5] https://sourceforge.net/projects/dex2jar/

[6] https://github.com/java-decompiler/jd-core

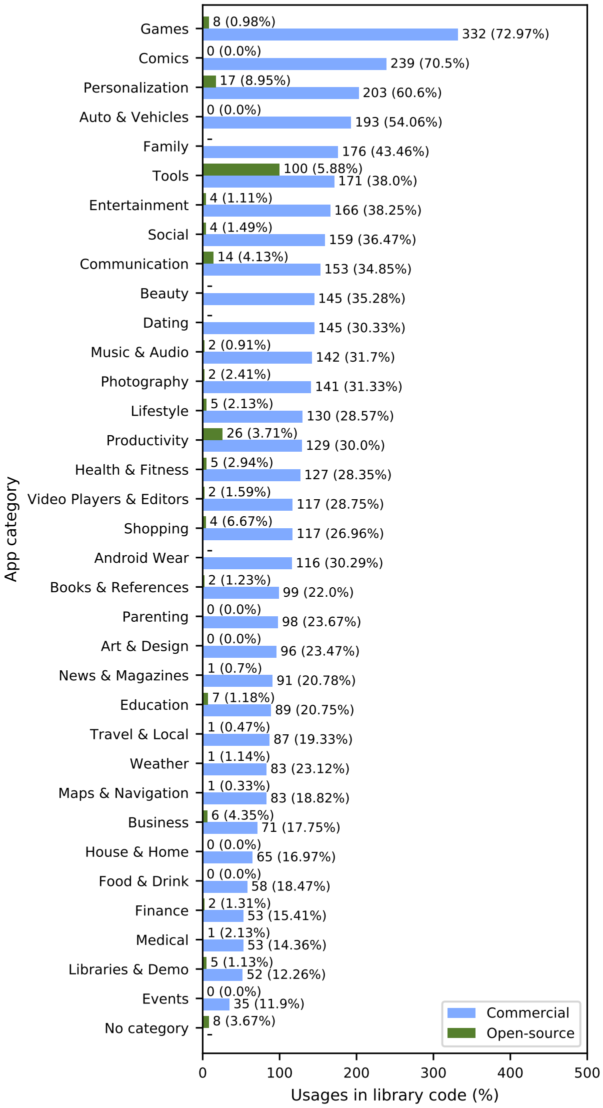

Figure 2: Usage of IAMs across Google Play Store categories

- Usage of IAMs is quite common in commercial apps, with an average usage rate of about 30% in our dataset.

- For commercial apps, adoption of IAMs greatly varies among app categories, being higher than 70% in the *Games* and *Comics* categories, while close to 12% in the *Libraries & Demo* and *Events* categories.

- Usage of IAMs is less frequent in open-source apps, with a 3% mean usage rate across categories in our dataset.

4.2 RQ2: What kinds of information are most frequently accessed through IAMs?

The results of the analysis of fields accessed with IAMs are displayed in Table 1. Surveying the table, it is evident that in both open-source and commercial apps *packageName* (i.e., the app name) is the most accessed field, being read by almost half of IAMs calls (47.62% and 46.90% in open-source and commercial apps, respectively).

Table 1: Access rate of IAM fields
(PI = PackageInfo, AI = ApplicationInfo)

Class	Field	Commercial (%)	Open-source (%)
PI	packageName	5502 (46.90%)	210 (47.62%)
AI	flags	1763 (15.03%)	42 (9.52%)
PI	versionName	706 (6.02%)	24 (5.44%)
PI	versionCode	678 (5.78%)	19 (4.31%)
PI	firstInstallTime	538 (4.59%)	7 (1.59%)
PI	lastUpdateTime	326 (2.78%)	7 (1.59%)
AI	sourceDir	259 (2.21%)	18 (4.08%)
AI	enabled	200 (1.70%)	6 (1.36%)
PI	receivers	132 (1.13%)	2 (0.45%)
AI	publicSourceDir	117 (1.00%)	3 (0.68%)
AI	uid	95 (0.81%)	9 (2.04%)
PI	providers	90 (0.77%)	3 (0.68%)
PI	requestedPermissions	78 (0.66%)	11 (2.49%)
AI	targetSdkVersion	63 (0.54%)	2 (0.45%)
PI	activities	56 (0.48%)	4 (0.91%)
PI	signatures	50 (0.43%)	1 (0.23%)
AI	processName	38 (0.32%)	1 (0.23%)
PI	services	20 (0.17%)	4 (0.91%)
AI	nativeLibraryDir	10 (0.09%)	0 (0.00%)
PI	sharedUserId	8 (0.07%)	0 (0.00%)
PI	CREATOR	8 (0.07%)	1 (0.23%)
AI	className	8 (0.07%)	1 (0.23%)
AI	dataDir	7 (0.06%)	3 (0.68%)
AI	theme	7 (0.06%)	0 (0.00%)
PI	permissions	7 (0.06%)	4 (0.91%)
AI	category	2 (0.02%)	0 (0.00%)
AI	manageSpaceActivityName	2 (0.02%)	0 (0.00%)
PI	reqFeatures	2 (0.02%)	0 (0.00%)
PI	gids	1 (0.01%)	1 (0.23%)
AI	permission	1 (0.01%)	6 (1.36%)
AI	descriptionRes	0 (0.00%)	1 (0.23%)
AI	sharedLibraryFiles	0 (0.00%)	1 (0.23%)
	Sum Total	11,732 (100%)	441 (100%)

Moreover, we can notice that the frequency of accesses to each field does not significantly differ among the two datasets, save for a few exceptions. The first of these exceptions is represented by *flags* (i.e., boolean flags about the app nature) that, while being the second most commonly accessed field for both commercial and open-source apps, appears to be less popular in the latter (9.52% as opposed to 15.03%). Other exceptions are represented by fields that contain information about application permissions (*permission*, *permissions* and *requestedPermissions*) and *uid* (the Linux kernel user-ID that has been assigned to the application) that in our data appears to be more often accessed by open-source applications.

Finally, we can observe that among the most frequently accessed fields, there are several ones which are related to app versioning and management of updates (*versionName*, *versionCode*, *firstInstallTime* and *lastUpdateTime*). As we will discuss in Section 5, this similarity in behavior between open-source and commercial apps hints that there exists a significant challenge for techniques that aim to protect end-users' privacy by selectively blocking undesired IAM calls. Synthesizing our findings:

• *packageName* is the information most frequently collected through IAMs, accessed by 47.62% and 46.90% of all IAMs calls performed in open-source and commercial apps, respectively.

• For the majority of fields that can be accessed with IAMs, the frequency of accesses does not significantly differ between open-source and commercial apps.

• Information about application permissions and *uid* appears to be accessed more frequently by open-source apps in our dataset.

4.3 RQ3: How are usages of IAMs distributed between app code and included libraries code?

The heatmap in Figure 3 summarizes the distribution of usages of IAMs between local and library code in our datasets. A total of 7,538 and 287 calls to IAMs were detected in commercial and open-source apps respectively (some apps perform more than one call). Usages of IAMs in included libraries appear to be more common in commercial apps, where 6,306 (83.66%) of detected calls are performed in code belonging to libraries, while the remaining 1,232 (16.34%) are performed in the apps' own code. Concerning open-source apps, 178 usages (62.02%) are performed from bundled libraries while remaining 109 (37.98%) belong to the apps' own code.

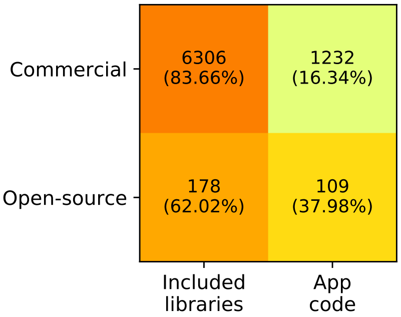

Figure 3: Usage of IAMs in app code and included libraries

Figure 4 presents the rate of usages in libraries for each category. Each bar in the plot provides the total amount of IAM calls found in included libraries (the remaining ones are performed in apps' own code) for applications of a given category. Categories for which no apps were collected or no IAM calls were observed are marked with the symbol "–". It can be noticed that, for commercial apps, in most categories IAMs appear to be used mostly in included libraries, with it being almost exclusive in categories such as *Comics* (98.47%), *Auto & Vehicles* (98.38%) and *Art & Design* (97.6%). A more even distribution of usages can be observed in a limited amount of categories, such as *Productivity* (59.24%), *Lifestyle* (52.12%) and *Business* (46.43%). Focusing in open-source apps, exclusive usage of IAMs methods in libraries can be observed in some categories, such as *Social* and *Shopping*. However, this trend can possibly be accounted to the low sample size of those categories. Indeed, categories *Productivity* and *Tools*, the ones with the higher amount of occurrences, exhibit a more even distribution with 68.75% and 56.00% of usages in included libraries respectively. To recap:

• In commercial apps, the vast majority of IAMs calls are performed in included libraries, representing 83% of all usages in our dataset.

• For commercial apps, usage of IAMs in libraries greatly varies among app categories, being almost exclusive inside the *Comics* (98.47%), *Auto & Vehicles* (98.38%), and *Art & Design* (97.6%) categories in our dataset.

• Also in the case of open-source apps, IAMs are mostly used in libraries, although in our dataset proportions are more even (62% in libraries against 38% in the app's own code).

4.4 RQ4: What is the declared main role that libraries calling the IAMs play?

Figure 5 provides the results of our analysis of the declared main role of libraries employing IAMs, following the procedure described in Section 3.4. Of the 154 analyzed libraries, the declared main role of more than one third (56) has been classified as *Advertising*. The role of roughly another one third (47) has been classified as *Utility*, grouping together libraries that serve varied purposes (e.g., push notifications, in-app billing, social networks integration) with the ultimate goal of streamlining app development. The remaining one third is composed as follows: 20 libraries are created in a custom manner by developers who want to enact interaction between other apps developed by them, 17 have an *Undetermined* purpose despite our efforts, 12 are *Analytics* libraries, and 2 focus on *App promotion*. The 17 instances of *Undetermined* role were mostly due to the use of obfuscation techniques, resulting in semantically meaningless package names.

Table 2 displays the 20 most popular library packages that employ IAMs in our combined datasets, with each respective number of occurrences. We can notice that 12 out of 20 have *Advertising* as their main role, including the top 6 most popular ones. These libraries have a large number of occurrences within our dataset (i.e., they are employed by a high number of apps), totaling 33.71% of all occurrences of libraries that employ IAMs. Of note, Unity3D, the most popular library, has been found to covertly read and upload to a remote server the user device MAC address in a recent study [32]. The remainder is composed of 3 *Utility* libraries, 2 *Analytics*, 2 *App promotion* and 1 whose role could not be determined. In short, our findings:

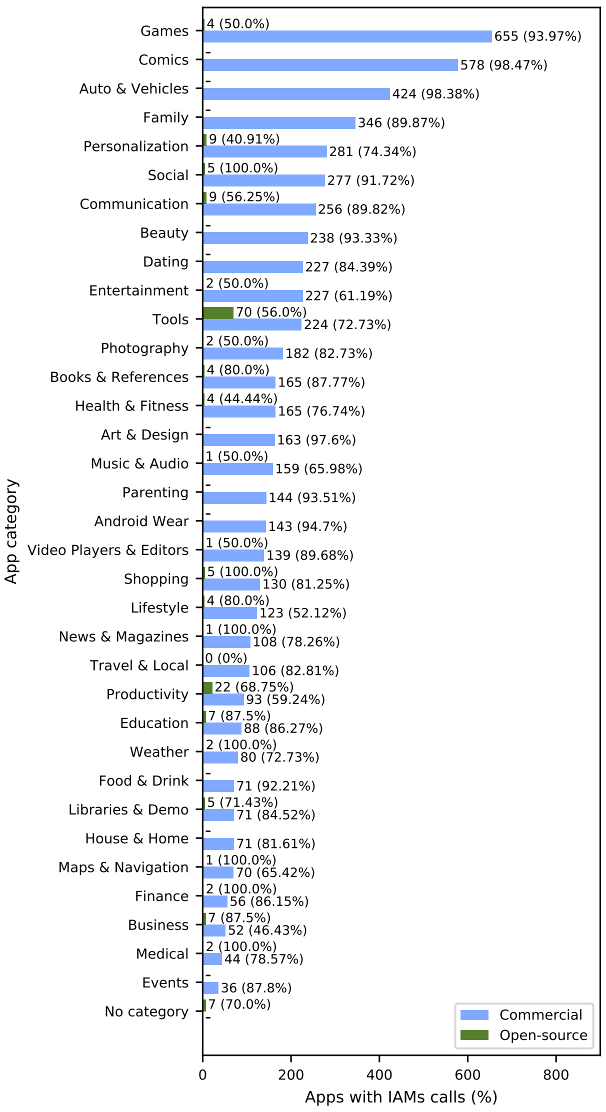

Figure 4: Library usage of IAM across categories

- More than one third of libraries that employ IAMs are *Advertising* libraries and roughly one other third are *Utility* libraries. The remainder is mostly comprised of *Custom libraries*, *Analytics* and *App promotion* libraries.

- *Advertising* libraries are by far the most diffused users of IAMs, with a small number of popular advertising libraries accounting for over 33% of all IAMs library usages.

4.5 RQ5: To what extent are developers aware of IAMs and tend to reflect that awareness?

A total of 72 participants completed the developer questionnaire, leaving us with 70 legitimate responses after discarding invalid ones.

Given the low number of replies, we will limit our quantitative analyses and only briefly report on answers to Q_1, as there is a considerable risk that collected answers might not generalize to the general developer population. Instead, we will focus more deeply on answers to the open questions Q_2 and Q_3, from which we can extrapolate qualitative insights. We believe that the low response rate is due to a combination of multiple factors: (i) as we will see later, the developers are generally unaware of IAMs usage in their apps and therefore might have disregarded the invitation email to answer our questionnaire; (ii) email addresses extracted from the Google Play store do not always provide a direct contact with the app developer(s) but might forward the mail to other stakeholders; (iii) the invitation email might have been blocked by spam mail filters.

With respect to Q_1, 47 developers answered that IAMs are used in core-functionalities of their app, 12 reported that they are used in included libraries, 7 declared that they are not used and 4 selected *"Other"* as an answer. Of the latter 4, one specified that IAMs are employed in his own framework, one that they are used in app functionalities that are not core, one that he was not aware of the presence of IAMs in his app and the last that he would need to check the code before providing an answer. Although the size of our sample is limited, it appears that while IAMs are often leveraged to provide some key app functionality there is a considerable amount of cases in which they are used exclusively by included libraries, sometimes unbeknownst to the developer itself.

The collected open responses (Q_2 and Q_3) reveal a clear lack of awareness of some developers regarding IAMs and their use in libraries. The 7 developers that declared that IAMs are not present in their apps confirmed to be oblivious about their presence, providing responses such as *"My app doesn't use this call."* and *"We don't at the moment."*. In addition, 9 out of the 12 developers that declared that IAMs are used by libraries in their apps were surprised that such methods were found inside their applications and they assumed they are used by included libraries. For instance one developer answered *"We were not aware that it was used at all. This API is likely called from an advertising library or from something like Google*

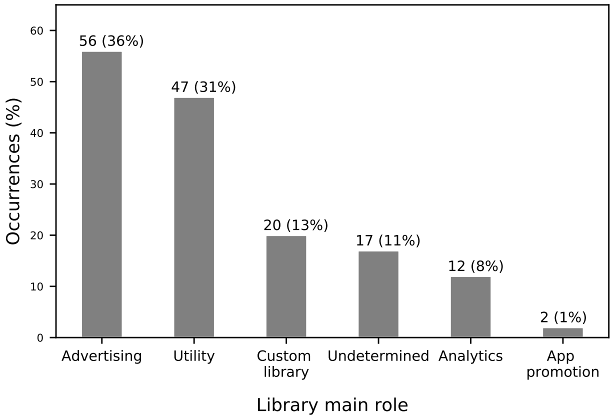

Figure 5: Assessed purpose of frequently used library packages

Table 2: Top 20 library packages using IAMs

Package	Occurrences (%)	Role
com.unity3d.ads.api	815 (10.81%)	Advertising
com.ironsource.environment	310 (4.11%)	Advertising
com.ironsource.sdk.controller	279 (3.70%)	Advertising
com.unity3d.services.core.api	234 (3.10%)	Advertising
com.google.ads.conversiontracking	175 (2.32%)	Advertising
com.appnext.base.b	158 (2.10%)	Advertising
com.pollfish.f	127 (1.68%)	Analytics
com.yandex.metrica.impl	127 (1.55%)	Analytics
com.tapjoy	125 (1.66%)	Advertising
com.heyzap.house	117 (1.55%)	Advertising
com.heyzap.internal	110 (1.46%)	Advertising
com.onesignal	97 (1.29%)	Utility
com.unity3d.ads.android.data	88 (1.17%)	Advertising
com.appodeal.ads.f	73 (0.97%)	Advertising
c.m.x.a.am48	70 (0.93%)	Undetermined
hotchemi.android.rate	69 (0.92%)	App promotion
io.branch.referral	68 (0.90%)	App promotion
com.google.zxing.client.android.share	64 (0.85%)	Utility
org.onepf.oms.appstore	61 (0.81%)	Utility
com.startapp.android.publish.common	57 (0.76%)	Advertising

Play Game Services. We use a variety of advertising SDKs within our game app.". Another stated *"We aren't using it. Third-party API? If you can tell me which one I'll remove it.".*

Replies such as the ones above clearly show a missing link where the developers are ignorant to the fact that libraries are using the IAMs in their applications. We assume that the black-box nature of code libraries is the main reason, as will be further discussed in Section 5. However, an answer in particular sheds some light on one of the reasons behind developers lack of awareness: *"The app is created on www.seattleclouds.com which is a drag and drop app builder. Contact them and ask them personally why they include this functionality.".* From it we can infer the existence of third-party app building services that introduce calls to IAMs unbeknownst to the app provider. This is consistent with the behavior observed by Watanabe et al., that noticed that these services tend to add unnecessary permissions or code [48].

One participant even showed resentment against library usage of IAMs, stating that, after learning more about reasons for their usage, he decided to no longer use the library in his applications: *"I recently found out about this call through the third party API, SafeDK. SafeDK reported that the InMobi library was polling the device for installed applications. This is NOT OK with me and I just uninstalled the InMobi library from all of my Android apps. I don't believe this call is needed or fair to users. I've notified InMobi that it's not OK to secretly poll devices for installed applications.".* Although this shows care for users, it also highlights how difficult it is right now for developers to know whether their used libraries make use of such methods or not.

Finally, developers who are actively using IAMs in their applications provide a wide range of reasons for their usage. IAMs are the basis for launcher apps, i.e., applications that allow for the customization of the home screen and provide shortcuts to launch other applications (reported as reason for usage by 9 participants). Similarly, IAMs are employed by applications that monitor or help manage other apps, e.g., Virtual Private Networks (3 participants), backup software (2), notification managers (2), anti-malware (1), battery savers (1), and firewalls (1). Notably, several developers report using IAMs to enhance the user experience. In particular, IAMs play a role in assisting users with disabilities: *"The user can select which apps they want our app to work in, or not. Note that our app is an accessibility service, so it is always active by default".* Other enhancements include providing improved social network sharing capabilities (2), managing passwords (1) and identifying installed web browsers to handle external urls (1). Finally, IAMs are used by developers to enable cooperation among multiple of their apps, e.g., one developer stated: *"We have an internal ads service, which shows ads for other applications written by our company. In order to show to the user ads for applications that aren't installed on his device, we use this API.".* Similarly, others pointed out interaction among multiple of their apps (4), the existence of theme or plugin systems (3) and their usage to verify the presence of the paid version of the app on the device (1). This diversity highlights that IAMs are key to a variety of applications and purposes, hence changes to the inner workings of these methods must be carefully thought through as they can potentially introduce breaking changes affecting a wide range of applications. One developer in particular expressed discomfort towards a possible deprecation of the IAM, stating that *"if Google deprecates this API I'll lose my mind.".* Summarizing our findings:

- Developers are not always aware of having included calls to IAMs in their apps, often introduced by enclosed libraries without their knowledge.
- Some app-builder frameworks and services automatically introduce IAM calls in the app.
- Some developers are against the use of IAMs for advertising purposes, to the point of having removed the libraries that employ them from their apps after becoming aware of their behaviour with respect to IAMs.
- IAMs are used for a variety of different purposes and some developers have spoken out against their potential deprecation or removal.

5 DISCUSSION

Our analysis reveals that IAMs are widely used in commercial applications, with a mean usage rate of about 30%, despite the fact that their diffusion greatly varies across app categories. Instead, popularity of IAMs is much lower in open-source apps, among which the mean usage rate is only 2.89%. IAMs appear to be used for varied purposes, to the point that some developers believe they are essential for their apps and expressed discomfort towards their possible deprecation. However, one of the main factors behind the popularity of IAMs is their usage by advertisement libraries: we identified 56 different ad libraries that rely on IAMs, with the most popular ones contributing for over 33% of occurrences of

IAMs-employing libraries. Notably, this finding is in contrast with previous studies: Grace et al. [20], in May 2011, observed that only 3 out of the 50 most used advertisement libraries made use of IAMs; Demetriou [11], in 2016, identified 28 ad libraries that invoke IAMs but reported that only 12.54% of apps in their dataset made use of such libraries. Although further data is required to be certain of the reasons for this discrepancy, we speculate it might be due to an increase in adoption of IAMs by advertisement companies, that over time learned to incorporate installed apps in their data collection practices. As a future work, we plan on investigating this point further, expanding our study to assess how IAMs' adoption has evolved over time.

The popularity of IAMs in advertisement libraries also suggests an explanation for the discrepancy in adoption between open-source and commercial applications: similarly to what observed by Demetriu et al. [11] who noticed a much lower usage rate of IAMs in paid apps as opposed to free ones, we can attribute it to the fact that open-source apps do not rely on advertising as much as commercial ones. As such, when choosing apps, we recommend privacy-aware users to (i) make use of existing app vetting services (e.g., VirusTotal [46]) to scrutinize apps for privacy risks prior to installation (ii) prioritize advertisement free-apps, choosing open-source apps when possible. Another potential reason for the observed difference in adoption of IAMs among the two datasets could be that in commercial apps synergies and dependencies between apps developed either by the same company or by partner companies are more common, thus resulting in a more frequent usage of IAMs. This is another point that we plan on investigating in future work.

Given the above, concerns regarding end-users privacy naturally arise: as previously discussed in Section 2, IAMs can be used to infer users' traits in detail and often without explicit consent or notification. As other privacy-sensitive parts of the Android platform are protected by app permissions [12], forcing developers to explicitly notify users before attempting access to these parts, begs the question on why IAMs are treated differently. Indeed, the European Union General Data Protection Regulation (GDPR) [7], generally regarded as the forefront in privacy regulations [47], considers *"online identifiers provided by their devices, applications, tools, and protocols [...]."* as personal data, for all purposes and means. Hence, an immediate possible solution would be the introduction of a new permission so to make IAMs treated equally to other sensitive resources. Not only it would make usage of IAMs more explicit to users, but it would also contribute in raising developers' awareness about libraries usage of IAMs, as they would need to manually add the permission in the app manifest and source code when including one of these libraries into their apps. As privacy-aware users are skeptical of apps that require many permissions [37], in turn it would discourage unwarranted use of IAMs. The privacy concern can also be addressed by restricting IAMs querying capabilities to a subset of apps that must be declared beforehand in the app manifest file, similarly to how these methods are currently handled in Apple's iOS[2]. Clearly, this would come at the expense of those applications for which it is necessary to access the complete list of installed apps in order to function properly (e.g., battery savers, app launchers). Hence, a middle ground would be to allow users to disable IAMs from the Android system settings: on the one hand this would allow privacy-aware users to consciously avoid using those applications in exchange for an increased privacy protection, on the other hand unconcerned users would still be allowed to take advantage of such apps on their devices.

In regards to these concerns, an open area of investigation for researchers is the development of novel solutions that, while allowing for apps' interoperability, ensure that end-users privacy is preserved. In this direction, as future work, we plan on investigating the usage of machine learning techniques for the automatic identification and selective filtering of IAMs calls performed for profiling purposes, similarly to what has already been done for Android permissions [29, 50]. The first step to reach this goal is understanding what features can be leveraged to perform the filtering. Relying on the library package name does not seem to be a long term solution, as it can be changed dynamically to evade such filters. Similarly, the results presented in Section 4.2 suggest that discriminating IAM calls based on accessed fields is also not helpful, as the frequency of fields accessed does not significantly vary between open-source and commercial apps, despite the different popularity that advertising libraries experience between those two groups.

On a broader scope, the issues that emerge from our analysis can be seen as consequences of the black-box nature of software libraries. It is a known issue that the tendency of developers to include third-party code in their projects, without prior verification of its content, can lead to the inclusion of vulnerabilities or, in the worst case, of malicious code [10, 33, 43]. Indeed, in recent history, prominent package repositories for the Python and JavaScript languages have been the target of attacks that aimed to exploit the popularity of widely-used libraries [28, 42]. Likewise, the introduction of libraries that covertly employ IAMs endangers end-users' privacy and thus it confirms the need for novel solutions for scalable discovery, analysis, and vetting of software libraries [53]. Our previous work [35, 36] provides a first step in this direction, enabling mobile users with the means to better control the purpose and extent to which their personal data is used.

6 THREATS TO VALIDITY

In the following we discuss the threats to validity of our study, referring to the categorization by Cook and Campbell [8].

Internal validity refers to the causality relationship between treatment and outcome [51]. In our study we relied on decompilation of Android binaries to perform the data extraction from commercial applications, as described in Section 3.3. Decompilers are not always able to rebuild the original source code with full precision. Moreover, some app developers are known to rely on obfuscation techniques to hinder decompilation attempts. When extracting the fields accessed with IAMs calls, we limited the extraction scope to the same file where the IAM call is found, due to the expensive computational requirements of more precise analyses. Moreover, during the extraction, only direct field accesses were considered, without taking into consideration accessor methods. Hence, we might have missed some IAMs calls, or some details thereof. We mitigated this threat by relying on consolidated off-the-shelf tools during the data extraction process and by considering a large number of apps in the initial subject selection. Moreover, extracted data is informative even if it represents a lower bound of actual IAMs usage.

In our study a manual procedure was employed to ascertain the declared main role of included libraries, described in Section 3.4. Like all types of manual analysis, there is the potential for mistakes during the procedure. To mitigate this threat, two different researchers performed this task independently. Their agreement level was measured with the Krippendorf alpha [24] and resulted satisfactory.

External validity deals with the generalizability of obtained results [51]. To ensure that our subjects are representative of the population of Android apps, we considered both open-source and commercial Android apps. For the former, we downloaded the 7,886 apps included in the Androzoo dataset that are still available. For the latter, we downloaded the top 14,342 apps in the United States across all categories of the Google Play Store, as ranked by AppAnnie. Since the apps are the top ranking apps of all categories, we can expect that they have a high number of users because they are ranked using a combination of number of downloads and aggregate user ratings. In our study, mostly due to budgetary concerns, we did not consider paid apps. However, free apps represent 75% of all Google Play Store apps and they are downloaded more often [17].

Construct validity deals with the relation between theory and observation [51]. The goal of our study is to understand how IAMs are used in practice, for the purpose of gaining insights on what measures can be introduced to increase end-users privacy protection. Although in our study we considered several aspects related to IAMs usage, others have not been considered (e.g., how is collected information used, if it is transmitted to a remote server, how frequently is the list of installed apps collected). Hence, our analysis might not have uncovered the more subtle issues.

Conclusion validity deals with the statistical correctness and significance [51]. In the discussion of results, we assumed that vast majority of the IAMs calls performed by advertisement libraries are for profiling purposes, and we therefore suggested some potential changes to the Android platform accordingly. This assumption is substantiated by existing literature [11, 20, 31, 32]. Moreover, we report in the paper example answers to our developer questionnaire that further confirm the nonessential nature of these calls.

7 RELATED WORK

Seneviratne et al. [38, 39] conducted the first investigation on user profiling via IAMs. Malmi et al. [25] and Frey et al. [16] conducted similar studies. Demetriou et al. have investigated the extent to which information provided by IAMs can be leveraged by advertising libraries [11]. A discussion of these works is provided in Section 2. Our work is based on theirs, aiming to provide a broader picture of diffusion and actual usages of IAMs. Zhou et al. combined information provided by several seemingly innocuous Android APIs, including IAMs, to infer sensitive information, such as users' gender and religion [54].

The privacy risks posed by IAMs are particularly critical in the context of mobile health apps, as their mere presence on the user device can reveal particularly sensitive information. To address this issue Pham and colleagues designed *HideMyApp* [30], a system that hides the presence of sensitive apps relying on user-level virtualization techniques. Experimentally evaluated, HideMyApp introduced a negligible performance overhead and was well received by end-users.

More loosely related to our work are studies that investigated potential techniques for the unique identification of end-users. *Browser fingerprinting* has historically been used by companies to uniquely identify website visitors [15]. In the context of Android, multiple studies have investigated potential techniques for *app fingerprinting*, leveraging network traffic patterns [9, 44, 45, 52], power consumption [5], memory footprints [22], and UI states [4].

Researchers have also focused on analyzing and understanding the behavior of advertising libraries, to assess the possible risks deriving from it. Grace and colleagues [20] investigated potential privacy and security risks posed by embedded advertisement libraries commonly used in commercial Android apps. Their analysis highlights that these libraries often engage in risky behavior for end-users' privacy, ranging from uploading sensitive information to remote servers to executing untrusted code downloaded from Internet sources. Besides, their results show that most existing ad libraries collect a wide range of private information, that in some cases includes the list of apps installed on the phone. Stevens et al. [41] examined the effect on user privacy of thirteen popular Android ad providers. From their analysis emerges that several ad libraries check for permissions beyond the required and optional ones listed in their documentation and employ an insecure JavaScript extension mechanism. Worryingly, they show that ad providers can identify and track users across applications.

Recently, Razaghpanah and colleagues [31] performed a large scale study aiming at understanding the actors involved in the mobile advertising and tracking ecosystem and their commonly employed practices. Their results show that these services can track users leveraging a wide range of device identifiers without providing visual clues inside the apps. By analyzing network traffic they show that, after collection, sharing users' data with subsidiaries and third-party affiliates is the norm. Although IAMs can be used for tracking purposes, they have not been considered in the above study and, to our knowledge, our work is the first that aims to provide a measurement of their adoption in the wild.

8 CONCLUSION

We conducted a large scale empirical study to investigate how IAMs are used in practice, analyzing their usage in 14,342 free Android apps published in the Google Play Store and 7,886 open-source Android applications. For this purpose, we performed an analysis that discovers the information that is extracted through IAM usage in these apps to gain insights on potential user privacy concerns. We further administered an online questionnaire to app developers to assess their awareness about the presence of IAMs in their apps. Based on the results of our analysis, we formulated some suggestions for improving the Android platform to increase developers' awareness and end-users' control over IAMs. Future work involve investigating the evolution of IAMs adoption over time and developing novel solutions that allow apps' interoperability while preserving the end-users' privacy.

REFERENCES

[1] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 468–471.

[2] App Annie. 2017. Spotlight on Consumer App Usage, Part 1.

[3] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Acm Sigplan Notices*, Vol. 49. ACM, 259–269.

[4] Qi Alfred Chen, Zhiyun Qian, and Z Morley Mao. 2014. Peeking into Your App without Actually Seeing It:{UI} State Inference and Novel Android Attacks. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 1037–1052.

[5] Yimin Chen, Xiaocong Jin, Jingchao Sun, Rui Zhang, and Yanchao Zhang. 2017. POWERFUL: Mobile app fingerprinting via power analysis. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[6] Michael L Collard, Michael J Decker, and Jonathan I Maletic. 2011. Lightweight transformation and fact extraction with the srcML toolkit. In *2011 IEEE 11th international working conference on source code analysis and manipulation*. IEEE, 173–184.

[7] European Commission. 2018. General Data Protection Regulation.

[8] Thomas D Cook, Donald Thomas Campbell, and Arles Day. 1979. *Quasi-experimentation: Design & analysis issues for field settings*. Vol. 351. Houghton Mifflin Boston.

[9] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. 2013. Networkprofiler: Towards automatic fingerprinting of android apps. In *2013 Proceedings IEEE INFOCOM*. IEEE, 809–817.

[10] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the impact of security vulnerabilities in the npm package dependency network. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 181–191.

[11] Soteris Demetriou, Whitney Merrill, Wei Yang, Aston Zhang, and Carl A Gunter. 2016. Free for All! Assessing User Data Exposure to Advertising Libraries on Android.. In *NDSS*.

[12] Android Developers. 2018. *Permissions overview*. https://developer.android.com/guide/topics/permissions/overview

[13] Android Developers. 2019. *PackageManager Description*. https://developer.android.com/reference/android/content/pm/PackageManager

[14] Android Developers. 2019. *Permissions overview*. https://developer.android.com/guide/topics/permissions/overview

[15] Peter Eckersley. 2010. How unique is your web browser?. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 1–18.

[16] Remo Manuel Frey, Runhua Xu, and Alexander Ilic. 2015. Reality-Mining with Smartphones: Detecting and Predicting Life Events based on App Installation Behavior: Research-in-Progress. *ICIS 2015 Proceedings* (2015).

[17] Inc Gartner. 2012. Gartner Says Free Apps Will Account for Nearly 90 Percent of Total Mobile App Store Downloads in 2012. http://www.gartner.com/newsroom/id/2153215.

[18] Franz-Xaver Geiger, Ivano Malavolta, Luca Pascarella, Fabio Palomba, Dario Di Nucci, and Alberto Bacchelli. 2018. A graph-based dataset of commit history of real-world android apps. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 30–33.

[19] Lisa M Given. 2008. *The Sage encyclopedia of qualitative research methods*. Sage publications.

[20] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. 2012. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 101–112.

[21] IDC IDC. 2019. *Smartphone OS market share*. http://www.idc.com/prodserv/smartphone-os-market-share

[22] Suman Jana and Vitaly Shmatikov. 2012. Memento: Learning secrets from process footprints. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 143–157.

[23] Klaus Krippendorff. 2004. Reliability in content analysis. *Human communication research* 30, 3 (2004), 411–433.

[24] Klaus Krippendorff. 2018. *Content analysis: An introduction to its methodology*. Sage publications.

[25] Eric Malmi and Ingmar Weber. 2016. You are what apps you use: Demographic prediction based on user's apps. In *Tenth International AAAI Conference on Web and Social Media*.

[26] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. 2013. An empirical study of api stability and adoption in the android ecosystem. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 70–79.

[27] Paul Mihas. 2019. Qualitative data analysis. In *Oxford Research Encyclopedia of Education*.

[28] npmjs.com. 2018. *Details about the event-stream incident*. https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident

[29] Katarzyna Olejnik, Italo Dacosta, Joana Soares Machado, Kévin Huguenin, Mohammad Emtiyaz Khan, and Jean-Pierre Hubaux. 2017. Smarper: Context-aware and automatic runtime-permissions for mobile devices. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 1058–1076.

[30] Thi Van Anh Pham, Italo Ivan Dacosta Petrocelli, Eleonora Losiouk, John Stephan, Kévin Huguenin, and Jean-Pierre Hubaux. 2019. HideMyApp: Hiding the Presence of Sensitive Apps on Android. Proceedings of the 28th USENIX Security Symposium.

[31] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. 2018. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. (2018).

[32] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 603–620.

[33] Jukka Ruohonen. 2018. An Empirical Analysis of Vulnerabilities in Python Packages for Web Applications. In *2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 25–30.

[34] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

[35] Gian Luca Scoccia, Ivano Malavolta, Marco Autili, Amleto Di Salle, and Paola Inverardi. 2017. User-centric android flexible permissions. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 365–367.

[36] Gian Luca Scoccia, Ivano Malavolta, Marco Autili, Amleto Di Salle, and Paola Inverardi. 2019. Enhancing Trustability of Android Applications via User-Centric Flexible Permissions. *IEEE Transactions on Software Engineering* (2019). https://doi.org/10.1109/TSE.2019.2941936

[37] Gian Luca Scoccia, Stefano Ruberto, Ivano Malavolta, Marco Autili, and Paola Inverardi. 2018. An investigation into Android run-time permissions from the end users' perspective. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. ACM, 45–55.

[38] Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. 2014. Predicting user traits from a snapshot of apps installed on a smartphone. *ACM SIGMOBILE Mobile Computing and Communications Review* 18, 2 (2014), 1–8.

[39] Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. 2015. Your installed apps reveal your gender and more! *ACM SIGMOBILE Mobile Computing and Communications Review* 18, 3 (2015), 55–61.

[40] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2008. *Guide to advanced empirical software engineering*. Vol. 93. Springer.

[41] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. 2012. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, Vol. 10. Citeseer.

[42] Victor Stinner. 2017. *Typo squatting and malicious packages on PyPI*. https://mail.python.org/pipermail/security-announce/2017-September/000000.html

[43] Jeffrey Stuckman and James Purtilo. 2014. Mining security vulnerabilities from linux distribution metadata. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE, 323–328.

[44] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2016. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 439–454.

[45] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2017. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security* 13, 1 (2017), 63–78.

[46] Virus Total. 2012. Virustotal-free online virus, malware and url scanner. *Online: https://www. virustotal. com/en* (2012).

[47] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* (2017).

[48] Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, and Tatsuya Mori. 2015. Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps. In *Eleventh Symposium On Usable Privacy and Security ({SOUPS} 2015)*. 241–255.

[49] Fengguo Wei, Sankardas Roy, Xinming Ou, et al. 2014. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1329–1341.

[50] Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. 2018. Contextualizing Privacy Decisions for Better Prediction (and Protection). In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 268.

[51] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers.

[52] Qiang Xu, Yong Liao, Stanislav Miskovic, Z Morley Mao, Mario Baldi, Antonio Nucci, and Thomas Andrews. 2015. Automatic generation of mobile app signatures from traffic observations. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1481–1489.

[53] Rodrigo Elizalde Zapata, Raula Gaikovina Kula, Bodin Chinthanet, Takashi Ishio, Kenichi Matsumoto, and Akinori Ihara. 2018. Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm JavaScript packages. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 559–563.

[54] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. 2013. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1017–1028.