Alde: Privacy Risk Analysis of Analytics Libraries in the Android Ecosystem

Xing Liu[1], Sencun Zhu[2,3], Wei Wang[1(✉)], and Jiqiang Liu[1]

[1] School of Computer and Information Technology,
Beijing Jiaotong University, Beijing, China
{xingliu,wangwei1,jqliu}@bjtu.edu.cn
[2] Department of Computer Science and Engineering,
The Pennsylvania State University, State College, PA 16801, USA
szhu@cse.psu.edu
[3] College of Information Sciences and Technology,
The Pennsylvania State University, State College, PA 16801, USA

Abstract. While much effort has been made to detect and measure the privacy leakage caused by the advertising (ad) libraries integrated in mobile applications (i.e., apps), analytics libraries, which are also widely used in mobile apps have not been systematically studied for their privacy risks. Different from ad libraries, the main function of analytics libraries is to collect users' in-app actions. Hence, by design, analytics libraries are more likely to leak users' private information.

In this work, we study what information is collected by the analytics libraries integrated in popular Android apps. We design and implement a tool called "Alde". Given an app, Alde employs both static analysis and dynamic analysis to detect the data collected by analytics libraries. We also study what private information can be leaked by the apps that use the same analytics library. Moreover, we analyze apps' privacy policies to see whether app developers have notified the users that their in-app action information is collected by analytics libraries. Finally, we select 8 widely used analytics libraries to study and apply our method on 300 apps downloaded from both Chinese app markets and Google play. Our experimental results request the emerging need for better regulating the use of analytics libraries in Android apps.

Keywords: Android · Analytics libraries · Privacy leakage

1 Introduction

According to the statistical result from AppBrain [6], the number of apps in Google Play has reached 2.1 millions. The sheer number of apps that are in the Google Play and the number of new ones added daily only show that the mobile app ecosystem has become a gigantic marketplace that is still expanding. It is thus becoming more and more difficult for app developers to make their apps stand out. Hence, it is increasingly important for developers to understand their users and make their apps better for the users.

R. Deng et al. (Eds.): SecureComm 2016, LNICST 198, pp. 655–672, 2017.
DOI: 10.1007/978-3-319-59608-2_36

Collecting and analyzing the interactions between users and apps help developers to get insights about their users' in-app actions[1] and learn more about their users' behavior. With the analysis results, developers study the actions their users have taken and understand how the users use their apps. They can find out what problems their users are experiencing, and then work out solutions to fix the problems. This process of collecting and analysis is very important to developers for enhancing the users' experience. Hence, almost every popular app contains code snippets to collect and analyze users' in-app actions. Some developers implement the collecting and analysis functions by themselves, while others implement these functions with the help of some third-party libraries. We call a third-party library that is used to collect and analyze the users' in-app actions as "analytics library".

Analytics libraries are similar to ad libraries in some aspects. For example, they both are integrated with the host app. Host app and the library share privileges and resources. They have the same Linux file access control permissions and Android permissions. Both analytics library and ad library require some permissions that may not be needed by the host app. Therefore, analytics libraries may cause security and privacy issues similar to that caused by ad libraries [12,15]. However, ad libraries do not require developers to do many settings. Take AdMob's banner Ads [1] as an example, developers only need to add an ad view in their apps and set up the corresponding ad unit ID [2]. Then the ad library will automatically request ads and displays them in the ad view. Developers do not care so much about the ads' content. Though ad libraries have provided some ad control APIs, many developers do not use them [9,21]. In contrast, when developers use analytics libraries to collect users' in-app actions, developers need to invoke some tracking APIs provided by the analytics libraries at locations they want [14]. For example, developers may invoke the tracking APIs to collect user's payment action after the user touches a payment button. In other words, what information to collect is set by developers. The more a developer wants to profile his users, the more tracking points he will set in his app.

After collecting users' in-app action data, analytics libraries send it to the analytics companies, which analyze the data and present some results to developers. Now, curiosities are aroused on what private information is leaked to analytics companies and to app developers through this data. This problem is exacerbated because analytics libraries may collect unique device information (IMEI, MAC, etc.) that can be used to link the information collected by different apps together to get a more comprehensive record of users' activities. However, previous studies only concern the information protected by Android permissions or information input by users (e.g., account number, password), therefore, they cannot answer this question. As a first step in the direction of answering this question, we explore the users' in-app actions collected by the analytics libraries

[1] "users' in-app actions" means the users' behaviors when they are using an app, such as opening the app, browsing different pages in the app, pressing a button in the app, etc.

integrated in the popular apps. To fulfill this goal, we design and implement a tool called "Alde" (Analytics libraries data explorer) which employs both static analysis and dynamic analysis to discover the users' in-app actions collected by analytics libraries. In the static analysis process, Alde performs a backward trace analysis based on the app's smali codes [17]. This backward trace analysis aims to find out what information is sent to these APIs. In the dynamic analysis process, we hook the tracking APIs to explore what information is sent to these APIs at the app's running time. After obtaining the users' in-app actions collected by the analytics libraries, we manually review this data to determine what personal information is leaked to the analytics companies. We also manually review the popular apps' privacy policies to check whether they notify the users about such data collection. We select 8 widely used analytics libraries for study and apply our method on 300 apps downloaded from both Wandoujia (a Chinese app market) and Google Play. The experimental results show that (i) analytics libraries can be exploited by malicious developers to collect users' personal information directly; (ii) some apps indeed leak users' personal information to analytics companies even though their genuine purposes of using analytics libraries are legal; (iii) users will be deeply profiled if analytics companies link the information collected from different apps, especially in China; (iv) developers seldom describe the use of analytics libraries in their apps' privacy policies even though they are asked to do so. In a summary, we make the following contributions in this paper:

- To the best of our knowledge, our work is the first research focusing on understanding information leakage caused by the users' in-app actions collected by analytics libraries.
- We design and implement a tool named "Alde" that is used to discover the users' in-app actions collected by analytics libraries.
- We apply our method on 300 apps downloaded from both Wandoujia and Google Play and reveal the data collected by the analytics libraries integrated in these apps.

The remainder of this paper is organized as follows. Section 2 describes the background of Android analytics libraries. Section 3 gives our system design and implementation. Section 4 describes the dataset that we use in this study. Experimental results and related work are given in Sects. 5 and 6, respectively. Section 7 concludes our work.

2 Background

2.1 Analytics Libraries

Analytics libraries are important tools that mobile app developers commonly employ in their apps. Through them, analytics companies provide mobile app developers well analyzed data that shows how the users are using their apps. To understand how an analytics library is embedded into an Android app, next we provide a simplified structural overview of the mobile analytics library through Fig. 1.

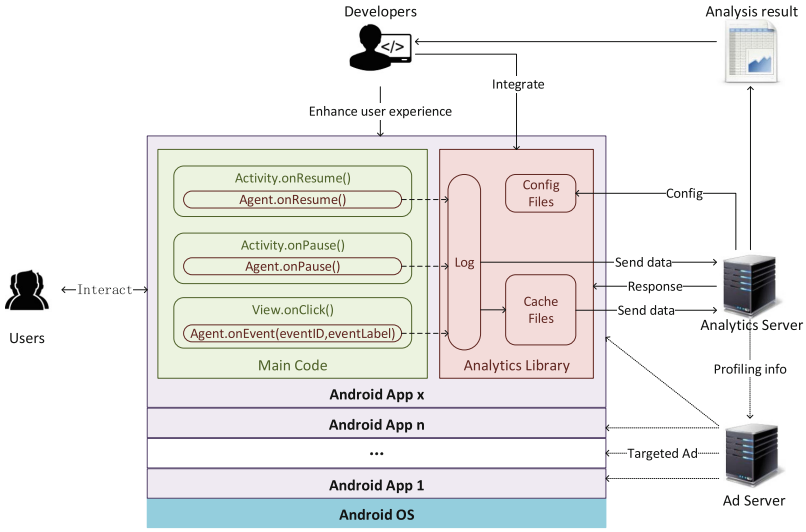

Fig. 1. Structural overview of a mobile third-party analytics library

Take Umeng [26], the most popular analytics library in China, as an example. In order to integrate this library into their apps and obtain the analysis results, developers need to take the following steps:

1. Register an account at the analytics company and log in. Then, a developer are required to set up the basic information (name, category, etc.) of the app that he wants to track. After the setup for the app, the analytics company will generate a unique AppKey. This AppKey will be utilized to track the app.
2. Add the SDK provided by the analytics company into the app's build path. Then, edit the app's AndroidManifest file and add the unique AppKey into the app's metadata. Moreover, the developer is required to add the permissions required by the analytics library into the AndroidManifest file.
3. Initialize the analytics library. Commonly, a developer needs to invoke the initialization method provided by the analytics library to initialize the library when the app is launched.
4. Invoke the tracking APIs provided by the analytics library to collect users' in-app action information. For example, with Umeng library, a developer can invoke *MobclickAgent.onResume()* and *MobclickAgent.onPause()* in each Activity's *onResume()* and *onPause()* methods to collect each Activity's start time and end time. He can invoke *MobclickAgent.onEvent(...)* to collect the users' in-app actions of his interest. For instance, if the developer wants to know how many users are interested in movies in a video app he developed, he can invoke this method (triggered when users press the "movies" button) and set the parameter *eventID* as "movies". Developers can also set up the

analytics library to automatically collect the run-time errors occurred in the app.

5. Upload the app to Android app market(s). When users download and enjoy this app, the analytics company will receive users' in-app actions data, analyze it and present the analysis results to the app's developer through a web interface.

The steps described above are the common procedures that developers need to follow if they want to use analytics library. Although most analytics libraries can be used successfully like this, different analytics libraries are different in implementation details. Hence, the processes of integrating different analytics libraries into the apps are not totally the same. Additionally, some new analytics libraries (such as *Appsee* [7] and *UXCam* [27]) use a totally different method to collect users' in-app actions. They do not require developers to invoke tracking APIs to collect users' actions. Instead, they collect all the interactions between users and apps as videos and show the videos to the developers directly. We do not consider this kind of analytics library in this paper.

2.2 What Information Is Presented to the Developers

When users play with apps, their in-app actions data is collected by analytics libraries and sent to the analytics servers. It is analyzed automatically in the analytics servers, which presents the analysis results to developers (See Fig. 2).

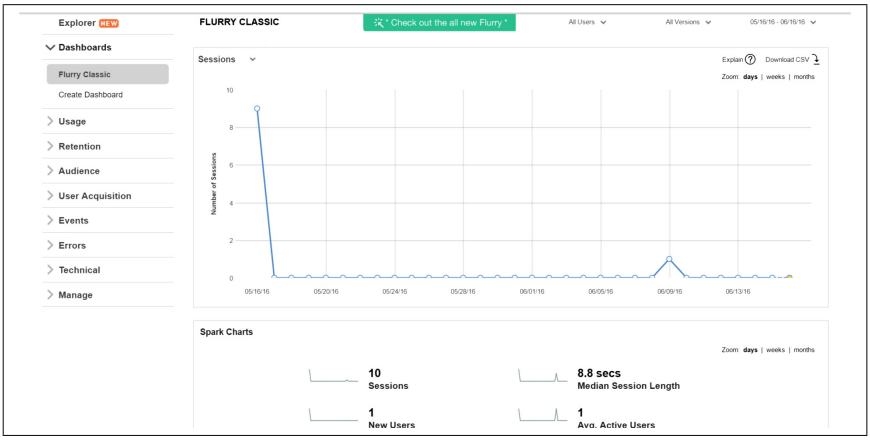

Fig. 2. Snapshot of flurry analytics

In Table 1, we list the information that developers can see. Besides the basic information shown in this list, analytics companies also present some statistical information, such as *User growth rate*, *User retention*, *User loyalty*, *Event conversion rate*, etc. This statistical information can be presented in different time

Table 1. The information that developers obtain about their apps

Categories	Details
Users	Total users, New users, Returning users, Active users, Launch times, Launch frequency, Duration of once use, Activity path, App versions
Terminals	Devices, Resolutions, OS versions, Carriers, Area, Languages
Events	Event IDs, Event labels, Event times, Event values
Errors	Error summary, Error times, First appearance time, Last appearance time

periods, by day, by week, by month, or by year, which helps developers learn whether their apps are popular or not in a period, or whether the new functions they added in the apps attract more users. Data presented to the developers is the statistical analysis results based on *all* users. In principle, developers cannot access the raw data of an individual user's in-app actions.

3 System Design and Implementation

To understand what private information can be leaked by the analytics libraries integrated with the popular apps, we develop "Alde", a tool for this purpose. Alde uses both static analysis and dynamic analysis to discover the values of the tracking APIs' parameters, which are the users' in-app actions collected by these tracking APIs. The overview of Alde is illustrated in Fig. 3.

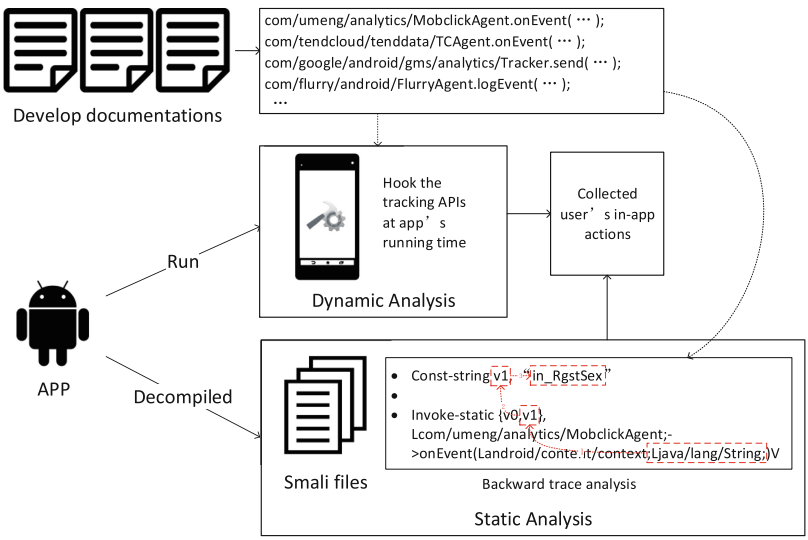

Fig. 3. Overview of Alde

3.1 Documentation Analysis

As we described in Sect. 2.1, developers need to invoke the tracking APIs provided by analytics libraries to collect users' in-app actions. Hence, our first step is to determine the tracking APIs provided by each analytics library. We obtain this information by analyzing the development documentations provided by each analytics library. However, some analytics libraries only give a brief description of the tracking APIs in their development documentations. The complete class names (including class package names) of the tracking APIs that are needed in the following processes are not given. To address this problem, we download some apps that contain these analytics libraries, decompile them with Apktool [5], and find out the complete class names of the tracking APIs in the decompiled codes.

3.2 Static Analysis

Some information collected by the tracking APIs is written in the app's source code, such as some buttons' names. Static analysis aims to discover the users' in-app actions defined in the app's source code. Alde performs a static backward trace analysis to find out the values of the tracking APIs' parameters based on the app's smali code. As shown in Fig. 3, given an app, Alde carries out the following analysis.

First, Alde decompiles the app into smali code files with Apktool. Second, Alde finds out the corresponding smali codes of the tracking APIs and identifies the registers that store the values of the tracking APIs' parameters. For instance, in Fig. 3, the second parameter of the *onEvent* method is the parameter that we need to trace. The corresponding register that stores the value of this parameter is $v1$. Third, Alde searches the smali code in the reverse order to find the value of $v1$. Alde decides what value is assigned to $v1$ based on the syntax of Dalvik bytecode [3]. If another register assigns its value to $v1$, that register will be traced instead of $v1$. This trace process will not stop until Alde finds a constant value is assigned to the traced register or Alde traces into a method that cannot be analyzed by Alde. Last, the final constant value and the trace path are reported. As shown in Fig. 3, the final constant value of $v1$ is "in_RgstSex".

The code snippet shown in Fig. 3 appeared in a fitness app. When users go to *Gender Setting* page, this code snippet will run and collect this in-app action.

3.3 Dynamic Analysis

Though the above static analysis can explore the in-app actions defined in app's source code, some information is generated at app's running time, so it cannot be captured by static analysis. Hence, Alde also performs a dynamic analysis on the app.

In the dynamic analysis process, Alde runs the app for 5 min with the help of AndroidViewClient [19]. Developed with python, AndroidViewClient is a test

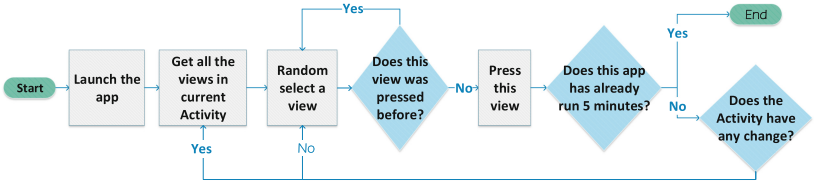

Fig. 4. Flowchart of the Alde's dynamic app running. The "views" in this figure include various elements in an Activity, such as buttons, text-areas, pictures, etc.

framework for Android apps and is more powerful than monkeyrunner [4]. We write a python script based on AndroidViewClient to automatically run Android apps. Given an app, Alde runs it according to the process described in Fig. 4. At the same time, the tracking APIs are hooked by Alde with the help of Cydia Substrate [22]. Cydia Substrate is an app running on rooted Android devices. It provides an easy way to hook the other apps running on the same phone. We develop an extension for Cydia Substrate to hook the tracking APIs. When the app under analysis invokes a tracking API, the values of the API's parameters will be captured by Cydia Substrate and stored in the files located in the phone's external sdcard. When the app stops running, we pull these files from the phone. Through this method, we get the users' in-app actions that are collected by the analytics libraries at the app's running time.

For the apps that ask the users to register an account, we register the account manually and then analyze it with Alde. After the entire analysis processes of an app are finished, we merge the analysis results from both static analysis and dynamic analysis to get the final analysis results (as shown in Fig. 5).

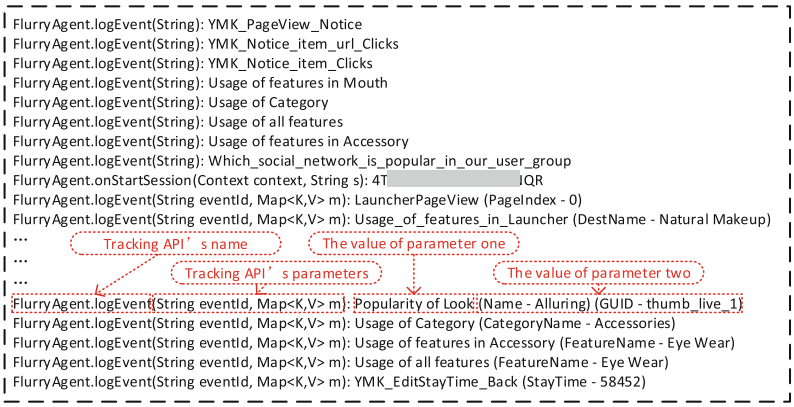

Fig. 5. Parts of the analysis result of app "YouCamMakeUp". If the parameter's type is *map*, the value of the parameter is presented as "(key - value)".

4 Dataset

In this section, we describe the dataset that we use in this study.

4.1 Analytics Libraries

In this paper, we focus on 8 widely used analytics libraries, shown in Table 2. To select these widely used analytics libraries, we search the Internet for analytics libraries and also learn from previous studies [20]. After this, we get a list of 25 analytics libraries. Then, we search these analytics libraries' class names in the smali code that is decompiled from the apps we downloaded. If an analytics library's class names appear in an app, we consider that this app uses this library. Finally, we select 8 most widely used analytics libraries in our app dataset (the rest of the analytics libraries in the list are seldom used in our app dataset). Four of them are mainly used by the apps in the Chinese app market and the other four of them are mainly used by the apps in Google Play. In the rest of this paper, we call them analytics libraries from Chinese app market and analytics libraries from Google Play, respectively.

Table 2. Analytics libraries' required permissions and optional permissions. "✔" means required permission and "●" means optional permission.

	Umeng	Talking data	Tencent analytics	Baidu analytics	Flurry	Adjust	Localytics	Google analytics
INTERNET	✔	✔	✔	✔	✔	✔	✔	✔
ACCESS_WIFI_STATE	✔	✔	✔	✔		●		
ACCESS_NETWORK_STATE	✔	✔	✔	✔	●			✔
READ_PHONE_STATE	✔	✔	✔	✔				
WRITE_EXTERNAL_STORAGE		✔	✔	✔				
WRITE_SETTINGS				✔				
GET_TASKS		●		✔				
READ_EXTERNAL_STORAGE				✔				
MOUNT_UNMOUNT_FILESYSTEMS				✔				
ACCESS_FINE_LOCATION		●		●	●			
ACCESS_COARSE_LOCATION		●			●			
BLUETOOTH				●				
WAKE_LOCK							✔	

Table 2 also shows the permissions required by these 8 analytics libraries as well as their optional permissions. Analytics libraries from Chinese app market commonly require more permissions. This is because they need the device information (IMEI, MAC, etc.) to generate the ID that is used to identify the individual device. And they also need to know the network state and WIFI state in order to adjust the interval of sending collected data to their servers. They may also need to store some cache files in the external storage. Meanwhile, analytics libraries from Google Play can do the similar things with the help of Google Play Service which is not available in China. However, these permissions

also give the analytics libraries from Chinese app market the abilities to collect more information than they need.

4.2 Apps

We download 200 apps from a Chinese app market[2] and 100 apps from Google Play. All these apps are popular and free apps. As described in Sect. 3, our method needs to know where the tracking APIs are invoked. If an app obfuscates the tracking APIs it used, we cannot apply our method on it. Hence, we run an API search (i.e., searching the tracking APIs' names in apps' smali code) to filter out the apps we can analyze. If a tracking API provided by an analytics library appears in an app's main package, we consider this app uses this analytics library and can be analyzed by our method. To understand how many apps are missed by our method, we carry out another file search process to determine the analytics libraries used by each app. In this file search process, we launch each app on a device and determine what analytics libraries it uses based on the files generated at the app's running time. This is because different analytics libraries will generate different files (such as database files, cache files, Shared_prefs files) at their running time. The generated files' names are not influenced by code obfuscation. We present the filtering result in Fig. 6.

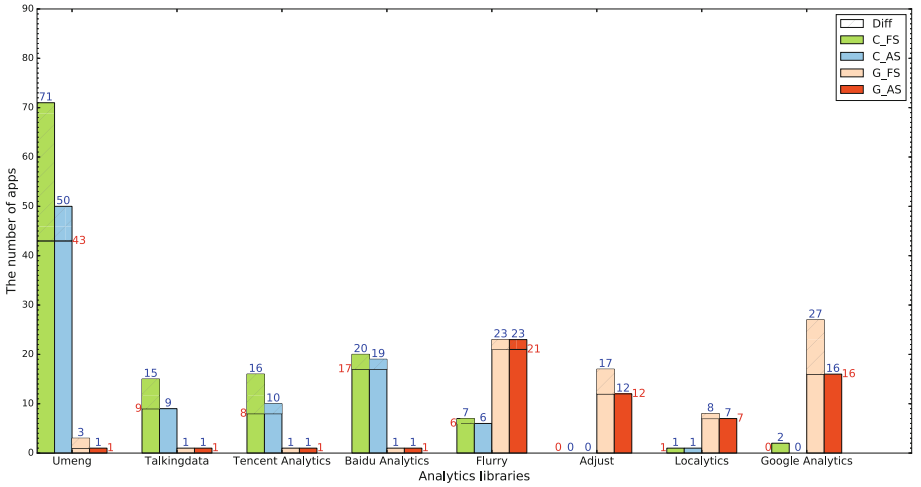

Fig. 6. The number of popular apps containing each of the analytics libraries. "Diff" means the apps on which these two kinds of search process generate different result. "C_FS" and "C_AS" means file search result and API search result of apps from Chinese app market. "G_FS" and "G_AS" means file search result and API search result of apps from Google play.

[2] We download Chinese apps from "Wandoujia" market. "Wandoujia" is a famous Android app market in China.

Figure 6 shows that our method can analyze most of the apps. We manually review the apps that obtain different results in these two kinds of search processes. We find that the apps not found in API search but found in file search indeed obfuscate the code of analytics libraries. More discussion on obfuscation will be presented in Sect. 5.3. Meanwhile, since dynamic running cannot cover all the code in an app, some apps do not generate the corresponding files at running time even they contain the tracking APIs. For such apps, we only consider their static analysis results in the following analysis. Finally, we select 81 popular apps from Chinese app market and 50 popular apps from Google play to analyze.

5 Experimental Results and Discussions

5.1 Experimental Results

We analyze the selected popular apps with our proposed method and then review the analysis results manually. Based on the information collected by the analytics libraries, we classify the apps into three levels: App level, Activity level and User level (See Table 3).

Table 3. The number of apps in each level of information collection

	Umeng	Talking data	Tencent analytics	Baidu analytics	Flurry	Adjust	Localytics	Google analytics
App level	8	1	4	4	9	5	2	4
Activity level	33	8	3	14	14	6	3	10
User level	9	1	4	2	6	1	3	2

In Table 3, "App level" means the app only uses analytics libraries to collect the information that reflects the running status of the whole app, such as what Activities are visited by the users. "Activity level" means the app uses analytics libraries to collect the running status of each Activity in an app, such as which "view" in the Activity is pressed by the users. A "view" means an element in an Activity, such as buttons, text-areas, pictures, etc. "User level" means the app uses analytics libraries to collect the data generated by the users. For instance, how long time a user spends on a song in a music app. Table 3 shows most apps belong to the Activity level.

In order to detail the results we found in our analysis, we organize them as the answers to the following four questions.

Q1: Do analytics libraries leak users' personal information to app developers?

As the developers cannot get the raw data of the collected information, it is hard for them to profile individual users. However, developers can exploit the vulnerabilities in these analytics libraries to collect users' private data directly.

For example, *Wo Mailbox* is a mailbox app that helps users manage their emails. It was developed by China Unicom and has more than 2.6 million active users in February 2016 [13]. Our tool finds that this app automatically records senders' email addresses, recipients' email addresses, email addresses of CCed recipients, emails' subjects and users' IP addresses through the analytics library.

We also find the analytics libraries do not check the information collected by the developers. They only perform some statistical analysis and present the analysis results to the developers. This makes it possible for developers to collect users' sensitive information through these analytics libraries. To test and verify this vulnerability, we developed two apps with Umeng and Talkingdata [25], respectively. We may disguise these two apps as communication apps, so it is reasonable for them to require READ_CONTACTS permission. When users open their contacts book with our apps, these apps read their contacts and show to them. Besides, these apps secretly collect their contacts through analytics libraries by invoking *MobclickAgent.onEvent(Context ctx, String eventId, Map eventValue)* for Umeng and *TCAgent.onEvent(Context ctx, String eventId, String eventValue)* for Talkingdata. Both Umeng and Talkingdata successfully collect users' contact information and present them to us through the servers' web interfaces (See Fig. 7 for the case of Umeng), although the tracking APIs we invoked are designed to collect user's in-app actions. Although we have not found real-world apps that have the similar behaviors, this vulnerability could be exploited for very stealthy information stealing.

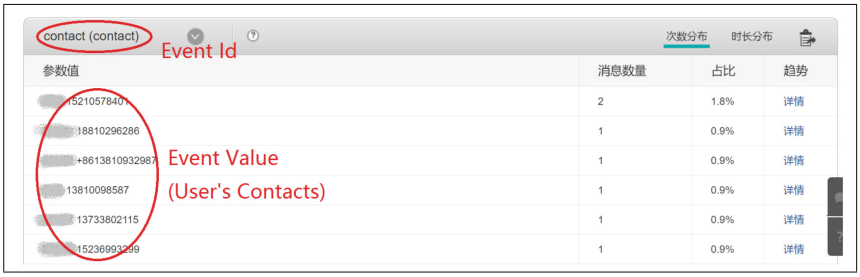

Fig. 7. User's contacts were successfully collected by Umeng

Q2: Do analytics libraries leak users' personal information to analytics companies?

Since analytics companies own the raw data of the collected information, compared with the information leaked to the developers, information leaked to the analytics companies is much more serious.

For example, *com.culiukeji.huanletao*[3] is a shopping app and *com.tadu.android* is a reading app. These two apps ask users to select their gender before using and collect user's gender information via Umeng. Their developers intend to understand the popularity of their apps in female or male users. They can get the percentage of female users and male users from Umeng. However, Umeng gets each app user's gender information through this way. Since Umeng collects the user's device identifier (IMEI, MAC, etc.) at the same time, it gets to know each device user's gender directly. *com.autohome.usedcar* is a used car trade app. It leaks user's fine location to Umeng. Apps in Google play also have similar behaviors. *Skype* sends call ended time and message sent time to Flurry. *Text Free* sends user's fine location, the rough number of the user's contacts and the rough length of every message to Flurry and sends the device's IMEI to Adjust. *The Weather Channel* leaks user's location to Localytics. Due to the space limit, we do not list all the apps that have the similar behaviors here.

Besides, some analytics libraries collect users' data secretly. Talkingdata is a well-known analytics library in China. We find that this analytics library reads the smartphone's sensors data without any notice to users and not even to developers. When developers invoke the tracking APIs provided by this analytics library to collect users' in-app actions, this analytics library will read the sensors' data (including ambient temperature sensor, relative humidity sensor, rotation vector sensor, pressure Sensor, light sensor and magnetic field sensor) and send the data to the analytics server. The collected sensor data will not be presented to the developers, and Talkingdata does not describe this behavior in their development documentation. Hence, neither the developers nor the users know about this. This is not a direct privacy risk, but this data indeed can be used to infer users' surrounding environment and sensitive information such as user touchscreen input [24,28]. During the course of our paper writing, Talkingdata released a special version of its SDK for Google Play, which has removed the code snippet for collecting sensor data.

Q3: What will analytics companies know about the users if they link the information collected from different apps?

As we mentioned before, the privacy risk caused by analytics libraries is exacerbated if analytics companies link the data collected from different apps together to profile the users. Analytics companies can do this work easily because they collect the device identifier together with the users' in-app actions. They know which apps are installed in the same device and used by the same user. The more popular an analytics library is, the more information it can collect. Take Umeng as an example, it is the most widely used analytics library in China. Apps integrating Umeng cover almost all the app categories (See Table 4). As these apps are popular apps, it is very possible that multiple of them are installed in the same phone.

[3] Some Chinese apps do not have corresponding English names, so we use their package names instead.

Table 4. App categories that Umeng collects data from

Category	Number of apps	Category	Number of apps
Health & fitness	1	Lifestyle	4
Photography	1	Tools	6
Weather	3	Music & audio	3
Media & video	10	News & magazines	1
Entertainment	2	Books & reference	3
Personalization	2	Finance	1
Travel & local	1	Communication	4
Education	4	Shopping	4

We review the information that is collected through Umeng to see what user's personal information may be inferred by Umeng if the user installs these apps. First, Umeng knows what apps that have integrated it are installed in the same phone. According to the previous study [12], this app install pattern will leak some user's information to Umeng. If the app is developed for particular users, more information will be leaked to Umeng. For example, *com.xtuone.android.syllabus* is developed for undergraduate students, *cn.haoyunbang* is developed for pregnant woman and new mother, and so on. Second, data sent to Umeng often has clear semantics. Umeng can learn user's gender and reading habits from a reading app; learn user's location and approximate income level from a used car trade app; learn user's video watching habits from a video app and learn user's health condition from a health app, etc. If Umeng analyzes and links all types of collected data, it can characterize the users in various aspects.

Q4: Do users know their in-app actions are collected by third-party analytics companies?

According to a previous study [16], 90% users care if apps share their personal data with third parties and 45% users believe the apps should never share their personal data with third parties without their explicit confirmation. This inspires us to see whether users know their in-app actions are collected by third-party analytics companies. Hence, we review these analytics libraries' privacy policies manually to discover what information they have claimed to collect. In these analytics libraries' privacy policies, we find that some analytics companies have listed what information they will collect and ask the developers to show the use of analytics libraries as well as the information collected by analytics libraries in their apps' privacy policies. However, after we review the privacy policies of the apps we selected, we find only a handful of apps follow this rule. In the 81 apps from the Chinese app market, only two apps clearly describe the using of third-party analytics services in their privacy policies, and only one of them gives the name of the analytics library it uses. In the 50 apps from Google Play, only

16 apps clearly describe the using of third-party analytics services and 4 apps give the names of the analytics libraries they uses. Hence, we believe most users do not know their in-app actions are collected by third-party analytics libraries.

5.2 Discussions

Our study shows the privacy risks stemmed from the analytics libraries. We think it might be caused by the following reasons.

First, in today's Android devices, users' private information is not limited to the information protected by Android permissions. Due to the lack of a clear definition on what information is users' personal information, developers might have difficulty in deciding what information should not be collected. Second, most developers disregard the end users' privacy, which can be known from the apps' privacy policies. Only a few apps describe the use of analytics libraries in their privacy policies. Third, some analytics companies do not provide privacy policies specifically for mobile analytics, which makes mobile app developers hard to understand the privacy risk caused by analytics libraries.

To protect users' privacy in this situation, we think the first thing is to let the users know what information is leaked through the analytics libraries in each app. Then they can choose to use the app or use another similar app. We believe that the app market should play the most important role. App markets can ask the developers to write clear descriptions about the using of analytics libraries and the information collected by analytics libraries in their apps' privacy policies. Our tool can be used by both users and app markets to explore the information collected by analytics libraries.

5.3 Limitations

In the static analysis process, Alde uses the methods provided by Apktool to decompile Android apps. Hence, we cannot analyze the apps that cannot be decompiled by Apktool. In the dynamic analysis process, we cannot cover all the execution paths. This is a common shortcoming of dynamic analysis. The most important shortcoming of our approach is that we cannot analyze the apps that obfuscate the tracking APIs they used. These limitations, however, may be overcome by our approach in the future. We can compare all the APIs in an app with the tracking APIs provided by each analytics library based on their instructions and call graphs rather than their names. In this way, we can identify which API is the tracking API even the API's name is obfuscated. Then we can use the obfuscated API name instead of the original API name in the following analysis.

6 Related Work

Privacy and Mobile Advertising. There are many studies focusing on the privacy issues associated with the advertising libraries in mobile apps. Grace *et*

al. [15] studied potential privacy and security risks caused by in-app ad libraries. They analyzed 100000 Android apps and found that most existing ad libraries collected private information. Book *et al.* [9] studied how app developers used the APIs through which a host app can send private information about the users to ad server. They found that although most apps did not make use of these privacy-related APIs, the number of apps that used these APIs is not negligible. The information collected by these APIs can be simply identified by the APIs' names. They [10] also studied mobile ad targeting using simulated user profiles and found that a large portion of mobile ads are targeted based on app, location, time, and profiles built around actual users. Nath [21] studied what targeting information was sent to ad networks by mobile apps and how effectively the information was used by ad networks to target users. Demetriou *et al.* [12] developed a tool called "Pluto" that can be used to analyze apps and discover whether they leak targeted user data. They also studied what ad networks can learn from the list of apps installed in a phone. Meng *et al.* [18] studied what ad networks know about the user's interest and demographic information. They also studied whether the host apps could conversely use the targeted ads to infer some of the user information collected by the ad network. Different from this studies, our study focuses on the analytics libraries.

Privacy and Mobile Analytics Service. Han *et al.* [16] studied how real-world users were tracked by the apps running on their Android smartphones. They employed dynamic information flow tracking to monitor when sensitive information was sent off the device. They recruited 20 volunteers to participate in this study. They found advertising and analytics were embedded in 57% of the apps and every participant in their study was tracked multiple times. However, they only studied the information protected by Android permissions. Chen *et al.* [11] studied the leakage of user's sensitive information through the vulnerabilities in mobile analytics services. They also studied how the ads served to users can be influenced by modifying the user profiles generated by these analytics services. Their experiments, conducted on Google Mobile Analytics and Flurry, validated the information leakage problem they described. They focus on the user profiles generated by analytics libraries and we focus on the tracking APIs.

Privacy and Mobile App's Privacy Policy. Slavin *et al.* [23] proposed a semi-automated framework to detect privacy policy violations in Android apps. They constructed a policy terminology-API map that linked policy phrases to API functions. Then they used this map to find the APIs and perform information flow analysis. They analyzed 501 top Android apps and discovered 63 potential privacy policy violations. But, they did not consider the information collected through tracking APIs. Yu *et al.* [29] developed a tool called "AutoPPG" that can be used to automatically construct correct and readable descriptions about the collection of user's private information. AutoPPG is able to generate the descriptions of third-party libraries used in apps; however, the information it focuses on is limited to the information protected by Android permissions. Balebako *et al.* [8] studied how app developers make decisions about privacy and security. They interviewed 13 app developers to get information about privacy

and security decision-making. And they test what they found with 228 app developers online. One important thing they found was that although third-party ads and analytics services are pervasive, developers aren't aware of the data collected by these tools.

7 Conclusion

In this paper, we studied the information leakage caused by analytics libraries that collect users' in-app action information. We developed a tool named "Alde" to explore the users' in-app actions. Through experiments on 8 popular analytics libraries and 300 apps downloaded from both Chinese app market and Google play, we found that some apps leaked users personal information to analytics libraries without notifying users. We also found that popular analytics companies have the capability to characterize and profile users. In the future work, we plan to improve our tool by making it more automated and more suitable for large-scale analysis. Then we will make it an online service to help users and app markets understand the information collected by analytics libraries.

Acknowledgment. We thank the anonymous reviewers for their insightful comments. This work was supported in part by the Scientific Research Foundation through the Returned Overseas Chinese Scholars, Ministry of Education of China, under Grant K14C300020, in part by Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, and in part by the 111 Project under Grant B14005. The work of Sencun Zhu was partially supported by NSF CCF-1320605 and CNS-1618684.

References

1. Admob: Admob by Google (2016). https://www.google.com/admob/
2. Admob: Admob quick start (2016). https://developers.google.com/admob/android/quick-start?hl=en
3. Android: Dalvik bytecode (2016). https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html
4. Android: Monkey runner (2016). https://developer.android.com/studio/test/monkeyrunner/index.html
5. Apktool: Apktool (2016). http://ibotpeaches.github.io/Apktool/
6. Appbrain: Appbrain stats, May 2016. http://www.appbrain.com/stats
7. Appsee: Appsee, 09 March 2016. https://www.appsee.com
8. Balebako, R., Marsh, A., Lin, J., Hong, J.I., Cranor, L.F.: The privacy and security behaviors of smartphone app developers (2014)
9. Book, T., Wallach, D.S.: A case of collusion: a study of the interface between ad libraries and their apps. In: Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, pp. 79–86. ACM (2013)
10. Book, T., Wallach, D.S.: An empirical study of mobile ad targeting. arXiv preprint arXiv:1502.06577 (2015)
11. Chen, T., Ullah, I., Ali Kaafar, M., Boreli, R.: Information leakage through mobile analytics services. In: Proceedings of the 15th Workshop on Mobile Computing Systems and Applications, p. 15. ACM (2014)

12. Demetriou, S., Merrill, W., Yang, W., Zhang, A., Gunter, C.A.: Free for all! Assessing user data exposure to advertising libraries on android. In: NDSS 2016 (2016)
13. Eguan: Eguan mobile apps top list, March 2016. http://qianfan.analysys.cn/user-radar/view/ranking/topRanking.html
14. Flurry: Custom events with flurry analytics for Android (2016). https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/events/android/
15. Grace, M.C., Zhou, W., Jiang, X., Sadeghi, A.-R.: Unsafe exposure analysis of mobile in-app advertisements. In: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 101–112. ACM (2012)
16. Han, S., Jung, J., Wetherall, D.: A study of third-party tracking by mobile apps in the wild. Technical report, University of Washington (2012)
17. JesusFreke: Smali (2016). https://github.com/JesusFreke/smali
18. Meng, W., Ding, R., Chung, S.P., Han, S., Lee, W.: The price of free: privacy leakage in personalized mobile in-app ads. In: NDSS Symposium 2016 (2016)
19. Milano, D.T.: Androidviewclient (2016). https://github.com/dtmilano/AndroidViewClient
20. Mobyaffiliates: The best app analytics tools list (2015). http://www.mobyaffiliates.com/guides/best-app-analytics-tools-list/
21. Nath, S.: Madscope: characterizing mobile in-app targeted ads. In: Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, pp. 59–73. ACM (2015)
22. LLC SaurikIT: Cydia substrate: the powerful code modification platform behind cydia (2016). http://www.cydiasubstrate.com/
23. Slavin, R., Wang, X., Hosseini, M.B., Hester, J., Krishnan, R., Bhatia, J., Breaux, T.D., Niu, J.: Toward a framework for detecting privacy policy violations in Android application code. In: Proceedings of the 38th International Conference on Software Engineering, pp. 25–36. ACM (2016)
24. Spreitzer, R.: Pin skimming: exploiting the ambient-light sensor in mobile devices. In: Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, pp. 51–62. ACM (2014)
25. Talkingdata: Talkingdata (2016). https://www.talkingdata.com/
26. Umeng: Umeng (2016). https://www.umeng.com/
27. UxCam: Uxcam, 09 March 2016. https://uxcam.com
28. Xu, Z., Bai, K., Zhu, S.: Taplogger: inferring user inputs on smartphone touchscreens using on-board motion sensors. In: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 113–124. ACM (2012)
29. Yu, L., Zhang, T., Luo, X., Xue, L.: Autoppg: towards automatic generation of privacy policy for Android applications. In: Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 39–50. ACM (2015)