What You See Isn't Always What You Get: A Measurement Study of Usage Fraud on Android Apps

Wei Liu[⋆†], Yueqian Zhang[⋆†], Zhou Li[‡], Haixin Duan[⋆†]
l-w14@mails.tsinghua.edu.cn,{zyq8709,lzcarl}@gmail.com, duanhx@tsinghua.edu.cn
[⋆]Tsinghua University, [†]Tsingua National Laboratory for Information Science and Technology, [‡]ACM Member

ABSTRACT

We studied a new type of fraudulent activities, usage fraud, on Android platform in this paper. Different from previous fraud on mobile platforms targeting advertisers or mobile users, usage fraud was invented to boost usage statistics on third-party analytics platforms like Google Analytics to cheat investors. To understand the business model and infrastructures employed by the fraudsters, we infiltrated two underground services, Laicaimao and Anzhibao. A number of insights have been gained during this course, including the use of emulators and manipulation of user identifiers. In addition, we evaluated the efficacy of the existing fraud services and the defense status quo on 8 popular analytics platforms. Our result indicates that the fraud services are indeed capable of crafting valid usage numbers and the basic checks are missed by analytics platforms. We give several recommendations in the end and call for the contribution from the community to fight against this new type of fraud.

1. INTRODUCTION

Android ecosystem is growing strongly in recent years and has attracted the majority of mobile users. Statistics show that Android dominates the mobile platform with the lion's share of 82.8% in 2016 Q1 [30] and nearly 2 million apps have been disseminated through the official Android store, Google Play [31]. Most of the developers and companies are striving to build high-quality apps, in the hopes of attracting more users and earning more revenue through advertisement or in-app purchase. A set of rules have been established by the industry leaders, e.g., Google, to regulate the behaviors of the participants, which however are unable to prevent ill-intentioned participants, or *fraudsters*, from violating the rules to gain profit in unfair manners. It has been discovered in the wild that fraudsters have frequently committed click and view fraud against advertisers [8] and manipulated app ranking in both official and third-party app stores [11], costing legitimate parties huge amount of loss [22].

The strategies and targets for fraudsters are constantly evolving. Despite the well-known ad fraud and app store fraud, a new type of fraud targeting app ecosystems called usage fraud surfaced recently, which attempts to tamper the usage statistics reported by the presumably trusted third-party *app analytics*. Usage statistics reflect how an app is treated by the users, e.g., whether it is constantly opened or just once, which provides much more finer-grained information than the statistics reported by app stores (e.g., the number of downloads and 5-star ratings). Hence, the reports from analytics services, like Google Analytics, have been seen as very important sources to judge the market value of the app, especially for investors. To meet the expectation of investors or attract new ones, dishonest developers start to use a variety of methods to forge the usage numbers. They could leverage underground services dedicated for such fraud or embed a snippet of code in their apps to trigger analytics APIs without consent from users. As an example, Qingting FM, one of the most popular online radio apps in China (10M+ installations), has been found to commit usage fraud through code embedding [1].

In this work, we made the first step to understand the new type of fraudulent activities and measured the defense deployed by third-party analytics. For the first task, we infiltrated two usage fraud services in China, named *Laicaimao* and *Anzhibao*, and used their infrastructures to promote our apps through controlled experiments. In particular, we created 8 apps packed with 8 SDKs offered by reputable analytics providers, uploaded them to Laicaimao and Anzhibao, and collected information about the devices running our apps and the results presented by analytics servers. Our preliminary results have brought up several meaningful insights about this underground business, including the wide adoption of off-the-shelf Android emulators like BlueStacks [5], substitution of device identifiers (e.g., IMEI) and their distributed network structures. Such fraudulent services turned out to be quite effective for boosting usage numbers. In fact, none of the analytics we surveyed were able to detect the presence of such services and remove the fabricated usage numbers from the report.

We further carried on to assess the defense from analytics platforms. Specifically, we looked into how users are counted by analytics platforms and what validation processes have been enforced by them. To this end, we performed an experiment by enumerating the identifiers in the network requests issued to the analytics servers and monitoring the responses. Surprisingly and disappointedly, even the basic checks (e.g., IMEI validity check) were neglected by the analytics and the obvious fraudulent patterns (e.g., regular transient sessions) were undetected.

Complementary to the study described above, we also looked into in-app usage fraud with code embedding. We extracted the signatures related to usage fraud through reverse-engineering Qingting FM (e.g., the use of invisible activity, remote services and broadcast receivers), leveraged them to build a customized code scanner, and screened over 10,000 Android apps. Though some apps were

SPSM'16, October 24 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4564-4/16/10. . . $15.00
DOI: http://dx.doi.org/10.1145/2994459.2994472

suspicious, none of them exhibit the similar type of fraudulent behaviors. As such, we conclude the in-app usage fraud is not a preferred option for adversary at present, probably due to its high risk of leaving evidence to users and security practitioners.

We believe analytics providers should take countermeasures against usage fraud to thwart attempts from the fraudsters and also avoid the damage to the credibility of their services. Fully mitigating usage fraud is very difficult at this stage, as the fraudsters can choose any device (or even emulator) with full control at their will, and learning the true user identities behind the devices is prohibited for the sake of users' privacy. Still, actions can be taken to raise the bar against fraudsters and we give several recommendations in the end.

To summarize, our main contributions in this study are threefold:

- *Understanding app analytics and their inherent issues.* We explain the mechanisms of app analytics and surveyed 8 popular app analytics on how they collect and process usage data. We describe the challenges in providing untampered usage statistics.
- *Measuring the underground business through infiltration.* Through running controlled experiments on two usage fraud services, we uncovered their infrastructures and the techniques they adopted. We also measured their efficacy in terms of users counted by analytics.
- *Systematic evaluation of different analytics platforms.* We ran "stress-tests" on the 8 analytics platforms to infer how they counted distinct users and the defense deployed by them.

2. BACKGROUND

We overview app analytics and their common metrics first. Then, we describe several types of known app fraud that aim to boost the volume of download, installation and usage. At the end of this section, we define the adversary studied in this work.

2.1 App Analytics

Mobile app developers need to understand who are using their apps and how the apps are used, in order to optimize app's user experiences and functionalities. To save the developers from building the component for performance analysis from scratch, there have been a number of companies like Google offering such services, which are called *app analytics*. Similar to its close relative, *web analytics* (or *web tracker*), app analytics consist of a client-side component which has to be included in app's code to collect the usage information per user and a server-side component which aggregates all collected information across users and reports them in different categories and metrics. Though an app analytics vendor could define its own measurement terms, there is a set of terms agreed upon broadly [33] and we describe them below:

1. **User.** A user is distinguished by the identifier which is either assigned by the mobile device (called device identifier) or generated by the analytics for the specific app (called analytics dedicated client identifier, or CID) , which we elaborate in Section 3. Data generated from different mobile devices under the same user is counted separately. The number of all acquired users is counted by all common analytics. In addition, the numbers of *new users* and *active users* are also measured, which indicate whether the app is continuously attracting users and actively used. A new user is counted when the app is launched for the first time on a device. When the app is uninstalled and reinstalled later, first-time launch will not be counted by analytics based on device identifier. A user is considered active when she opens the app days after it is installed.
2. **Session.** A session covers a period when an app is opened and terminated by the user. How session is interpreted differs in app analytics due to the different lifecycle management strategies enforced by mobile platforms (i.e., iOS and Android choose different ways in terminating apps). Average session length and page views per session across all users are usually reported.
3. **Page view.** A page view is counted when the user activates an in-app UI during a session. It usually refers to `View` in iOS and `Activity` or `Fragment` in Android. Average page views per session is reported by many app analytics.
4. **Event.** In addition to page view, an app developer could define custom events and make them tracked by app analytics. Examples include viewing photo, purchasing items, etc. Events can be reported in total, averaged per day or per session.

The vendors of app analytics usually provide SDKs to help developers integrate their services. As the starting step, the developer needs to register an account in analytics' websites and create a project which references the app by the package name. Next, the SDK (usually in a jar file) has to be included in app's package and the users' data will be automatically collected by invoking the APIs of the SDK when an activity starts or stops. The data is transfered to analytics servers and aggregated to be displayed to developers through web interface.

As a growing trend, besides app developers, the indexes presented by app analytics are also becoming important sources for investors to determine whether the app worths investment [34, 2]. As written in [34], analytics numbers are frequently seen in investor pitches by app developers. Naturally, apps showing good numbers in analytics are more favored by investors, which unfortunately motivates dishonest developers to produce fake statistics and cheat investors. Such fraud is actually happening. Recently, even very popular apps were caught for statistics cheating. As an example, Qingting FM, one of the most popular Radio apps in China market with 10M+ installations (counted by Wandoujia, a popular third-party Android market in China [36]), was found to trigger activities and invoke analytics APIs even when the app is not activated, in order to generate fake number of active users [1]. We consider this analytics fraud a big security issue as the integrity of analytics' data collection process is tampered, affecting the credibility of the platforms and benefit of investors. We carried out the first comprehensive study over this issue.

2.2 Fraudulent App Promotion

We consider app analytics as one particular type of fraudulent app promotion. In fact, there have been extensive reports exposing fraudsters who leverage different channels and techniques to promote apps in unethical ways, including manipulating ranks in app store (e.g., Google Play and iTunes) charts and installing apps when users are not aware. Below we elaborate the known techniques that support these types of fraud.

Phone/Emulator Farms.

Achieving high rank in app store charts is critical to the success of an app. While the ranking metrics are kept as top secrets by app stores, the number of downloads is considered as the major factor by the community doing App Store Optimization (ASO) [23]. The official stores like iTunes and Google Play all require valid user id and device id to be presented for a valid download event, and detects bulk downloads from a single source. To evade identity and source screening, fraudsters have built farms consisting of real phones or emulators and assign each unit with distinct user id and IP address. The download is carried out either by program or hired humans. Such downloading farms have become a real business globally and are widely used by app developers in China (called shuabang [11]) for app promotion. Knowing their existence, app stores are frequently upgrading the ranking algorithm and actively rooting out such services. Nowadays, it is still possible to manipulate the rankings, but the price is much higher (80K RMB or 12K US Dollars to rank top 3 in iTunes Paid App Chart [27]). To notice, such infrastructure can be also used to fake installation and usage statistics and we describe the real-world farms used for usage fraud in Section 4.

Fake Reviews.

Another important factor that decides app's ranking is the review in app stores. Many review services have emerged from which an app developer could buy a bulk of good reviews [38]. Such service recruits human writers and asks them to submit reviews manually using pre-registered accounts. To prevent the fake reviews from poisoning the rankings or misleading users, app stores like iTunes are actively detecting and removing false reviews [7].

Reseller and Repair Shop.

Despite the official stores managed by phone vendors like Apple and Samsung, users could go to third-party reseller and repair shops to buy and mend phones. It has been uncovered that these shops could affiliate with app promotion campaigns [39] and the staffs are encouraged to install apps on users' phone without consent. The staff earns commission fee for each app installation.

OEM or Rooted Phones.

To extend the utilities provided by native mobile OS, especially Android, OEM vendors like Samsung choose to customize the native OS and change the default settings. However, there are also vendors planting backdoors on the manufactured phones to later install apps without users' permission. Micromax, an OEM company in India, was caught disabling the default Google OTA service and using its own updating routine to silently push apps to customers' phones [12]. Rooted phone is another vector for promoting apps. Users could run root exploits to obtain full privileges on mobile OSes and bypass restrictions set by carriers. Previous research showed that root exploits could lead to severe security risks [40]. Similar to the OEM case, the root providers could also plant backdoors on the mobile devices and install apps silently. To make the behaviors stealthier, the apps could be installed when users are asleep and be removed automatically before users wake up [6].

ISP Hijacking.

All the Internet traffic from mobile users to app stores has to go through ISPs, including the traffic leading to app downloads. Such traffic can be manipulated by ISPs to download the apps they want to promote. It has been discovered by security practitioners [24] that the top ISP in China, China Telecom, replaced URLs pointing to Gmail Inbox app with URLs leading to other apps. In addition, a malicious app, YiSpecter, was also found to be distributed through ISP hijacking [37]. The best way to protect users from ISP hijacking is to enforce encryption over transport layer (e.g., deploying SSL), which however were still missed by many third-party app stores.

Embedding Fraud Code. As described in Section 2.1, dishonest app developer can force the execution of analytics APIs for usage fraud. Taking the case of Qingting FM as an example [1], the app runs a background thread which invokes a hidden activity without user interface and closes after two seconds every day. Since the hidden activity invokes analytics API automatically, the session is counted and the user is considered as an active user, even though she does not even open the app. Several analytics platforms, like UMENG and TalkingData, were defrauded in this case.

2.3 Scope

In this work, we focus on measuring the usage fraud happening in the wild. As far as we know, such fraud can be executed with the help of phone/emulator farms or through code embedding. The results regarding these two types are described in Section 4 and Section 6.

We only touched Android apps in this study, i.e., by using the fraud services to promote our test Android app and scanning off-the-shelf Android apps for fraud behaviors. The same issue might exist in iOS ecosystem but we think such fraud is less likely to be carried out, which we discussed in details in Section 8.

3. UNDERSTANDING APP ANALYTICS

Not only the statistics reports provided by app analytics can guide the developers to improve the design and implementation of their apps, they are also extensively used by investors to assess the value of the apps. Comparing to the coarse-grained data provided by the app store, e.g., the number of downloads, the data from analytics is more informative which tells how involved the users are. Though app developers could report the usage statistics using their own logging and reporting infrastructure, such data is less trustworthy. As such, third-party analytics platforms, especially the established ones, turn out to be the ideal choice for the report.

In this section, we first overview the app analytics regarding how and what information is collected. As showcase, we surveyed eight prominent app analytics, including Google Analytics, Flurry, Mixpanel, Localytics, UMENG, Tencent Analytics, Baidu Analytics and TalkingData (details listed in Table 1), which are all well recognized [33, 41]. Though these platforms are striving to provide authentic information regarding app usage, we found such guarantee is hard to meet due to the limitation of current mobile platforms, and we highlight the underlying issue. In the end, we describe other security and privacy issues we identified during our analysis of the analytics SDKs.

3.1 Mechanisms

A typical app analytics platform consists of a client-side component (i.e., SDK) integrated by apps and back-end service that merges the data across monitored user devices and reports it to the customers, i.e., developers and investors. Figure 1 illustrates the architecture and below we describe how usage data are collected, transmitted and aggregated.

Data Collection. Analytics SDK collects the user's identifiers and app's usage information and automatically transmits them to analytics server. To minimize the privacy risk to the app user, the best practice to identify a user is to generate a random value for her and use it throughout app's whole lifetime after the app is installed. However, this is only adopted by two analytics we surveyed, since such identifier is inconsistent when the app is uninstalled and reinstalled. The majority of analytics choose built-in device identifiers, including IMEI (or International Mobile Equipment Identity, a unique identifier associated with the GSM/LTE device), ICCID (or Integrated Circuit Card ID which distinguishes SIM cards), device MAC Address, Android ID (an ID set by Google when the device is booted up for the first time [16]) and GAID (or Google

Table 1: Surveyed app analytics.

Analytics	Version	URL	Main Market
Google Analytics	9.0.8	https://analytics.google.com/	Global
Localytics	3.4.0	https://www.localytics.com/	Global
MixPanel	4.8.0	https://mixpanel.com/	Global
Flurry	6.2.0	https://www.flurry.com/	Global
UMENG	5.6.4	https://i.umeng.com/	China
Tencent Analytics	2.1	http://mta.qq.com/	China
Baidu Analytics	1.4	https://mtj.baidu.com/	China
TalkingData	2.2.7	http://www.talkingdata.com/	China

Table 2: Identifiers collected by the analytics SDKs.

Analytics	IMEI	AID[1]	MAC	ICCID	GAID	CID[2]
Google Analytics	—[3]	—	—	—	—	√[4]
Localytics	—	√	√	—	√	—
MixPanel	—	—	—	—	—	√
Flurry	—	√	—	—	—	—
UMENG	√	√	√	√	—	—
Tencent Analytics	√	—	√	—	—	—
Baidu Analytics	√	—	√	—	—	—
TalkingData	√	√	√	√	√	—

[1] AID: short for Android ID
[2] CID: short for Analytics-dedicated Client ID
[3] —: The identifier is not collected by the analytics SDK
[4] √: The identifier is collected by the analytics SDK

Advertising ID, assigned by Google Play service [14]). The first three are persistent while the latter two could be reset when the device is restored to factory setting (Android ID) or reset by the user (GAID). Though using persistent identifiers for tracking purpose is forbidden by Google, we found there are 5 analytics ignoring such rule. We speculate such choice is made to accommodate countries where Google Play is blocked (e.g., China) [28] or devices running customized OSes. To access persistent identifiers, SDK has to ask for certain permissions (e.g., `READ_PHONE_STATE` for IMEI before Android 6.0 and `ACCESS_WIFI_STATE` for MAC) which are usually granted by the users.

During the lifetime of the monitored app, the usage information regarding the session, page views and events are collected. To track session, the developer is required to trigger the analytics APIs at the event handlers in `Activity` class, like `onResume` (the start of session) and `onPause` (the end of session). The interval of the session is computed by SDK as session duration. In addition, developer could call the analytics APIs at other event handers to record page views and customized events. To this end, the developer must specify the name of the viewed page (e.g., Home or Login) and describe user's action (e.g., in-app purchase).

To improve developer's visibility into app usage, auxiliary information like app's version and device model are usually collected by analytics as well.

Data Transmission and Aggregation. How the collected data is transfered to the analytics server varies per analytics provider. Most of the analytics SDKs aggregate the data and send them out in a batch regularly. For instance, Google Analytics sends the data every two minutes after the app is launched. There are also analytics SDKs sending out data once a monitored event is triggered, e.g., when the app is opened or a button is clicked. These two strategies are sometimes combined to maximize the transmission efficiency while at the same time keep the result up-to-date on the server. The data should be encrypted before sending out, but as elaborated in Section 3.3, certain SDKs fail to meet this basic requirement.

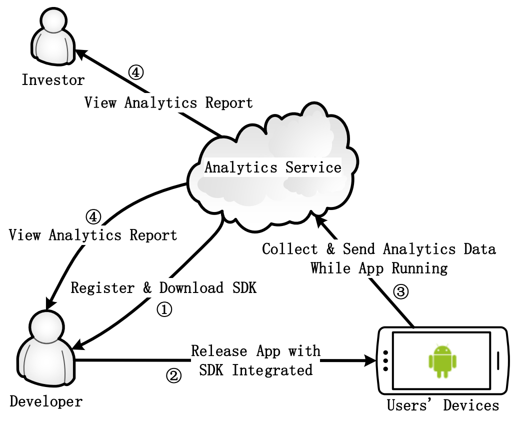

Figure 1: Architecture of the analytics platforms.

To separate the usage data between apps, an app key is generated when the developer registers her app on the analytics web site and should be attached to the usage data before it is sent out. The data across users are aggregated on the server and cataloged per app base. Analytics data are presented in variant ways. For instance, Flurry presents the number of new users and active users observed each hour/day/week/month. The data is further segmented by app's version, device model and user's location to help developer understand the audience.

3.2 Problem

In the beginning, app analytics are invented to serve the developers and assist them in improving the apps. The usage data is always considered organic as manipulating the data does not benefit any party (i.e., developers and users). Such premise is unfortunately invalid after the analytics reports are continuously reviewed by investors. The data transmitted to the analytics servers is a collection of user's identifiers, user-app interactions and auxiliary information (e.g., app key and device model). However, none of them are free from manipulation by motivated adversary, i.e., app developers or fraudulent services. Below we elaborate how these information can be maneuvered.

Faked Identifiers. The analytics platforms use the number of unique identifiers to compute user-centric metrics, like new users and active users. To learn which identifiers are used by analytics, we analyzed the network traffic originated from SDKs to search for fields resembling identifiers. Such process is not trivial as encryption and encoding are usually applied by the SDKs. To this end, we installed the certificate of Burp Suite [26] into our test device to intercept the traffic. We decrypted the post body by reverse-engineering the code of SDKs and managed to recover the connection between the post parameters and the identifiers. In the end, we were able to learn the list of used identifiers for all eight analytics.

As shown in Table 2, two out of the eight surveyed analytics leverage analytics-dedicated client ID or CID for short (Google Analytics and MixPanel) while the remaining ones use built-in identifiers. CIDs are easy to fake when knowing the format defined by the analytics providers. Producing false Android ID and GAID is also a trivial task and there is no reliable way to validate them, because Google has intentionally unlink them from devices to protect user's privacy. In fact, a recent report revealed that fraudsters could gener-

Table 3: Security and privacy issues with analytics SDKs.

Analytics	Collection	Communication
Google Analytics	—	HTTPS
Localytics	—	HTTPS
MixPanel	—	HTTPS
Flurry	—	HTTPS
UMENG	—	HTTP
Tencent Analytics	SSIDs	HTTP+RC4
Baidu Analytics	—	HTTP+AES
TalkingData	SSIDs App List	HTTP

ate massive Android IDs for in-app purchase fraud through frequent device resetting (called "mobile device flashing") [10]. For persistent identifiers, the manipulation is still feasible. MAC addresses can be faked on rooted devices or even temporarily replaced without rooting [35]. Though there are websites [19] providing IMEI validation services, they are only able to verify the format of IMEI or whether the associated device is stolen. Such check is easy to pass if the crafted IMEI is fresh and adheres to the format. For ICCID, the check is similar [18]. The device manufacturers and carriers can verify if the IMEI and ICCID belong to real devices or even identify the user behind the device, but asking them to share such information might be impractical and would be intrusive to user's privacy.

Invoking Analytics APIs without User Interaction. A fundamental issue with usage logging is that such operation has to be carried out by the app, which is under the full control of the adversary we studied here. As described in the Qingting FM example, logging functions are invoked regularly by the adversary to simulate user interactions even when the app stays in the background. What's worse, the fraudulent app could download the code from remote server on the fly and remove it after execution, making evidence collection much more difficult. As a countermeasure, analytics SDK could leverage the trust computing base (TCB) provided by device manufacturer (e.g., ARM TrustZone) to certify user interactions. Such idea has been explored as a solution against ad fraud on Android devices [20]. Still, this solution requires modifications to hardware and mobile OSes, which cannot be deployed broadly in a short time.

Emulators and Rooted Devices. The analytics SDKs rely on the system APIs provided by Android to retrieve identifiers and auxiliary information. The integrity of the system APIs can be easily compromised if the adversary uses emulators and rooted devices. Adversary can intercept the system calls and return spoofed identifiers and device information. As already discovered in the wild, the fraudsters can create a list of spoofed identifiers and randomly select one for in-app purchase fraud using instrumented API, which defeats detection mechanisms looking for excessively used identifiers [10]. For SDK, it is very difficult to determine if it runs on such environment as all the channels between OS and it are hooked by the adversary.

3.3 Other Issues with Analytic SDKs

During the course of our study, we also discovered several security/privacy issues with the SDKs. Some of the issues have also been discovered in mobile Ad SDK [17, 28, 13, 32]. Due to the vast amount of installations of these analytics SDKs, we believe these issues should also be addressed as soon as possible. These issues are elaborated below and summarized in Table 3.

Excessive Information Collection. To protect user's privacy, analytics SDKs should refrain from collecting information that can identify the user. However, it turns out such practice is not well followed. Besides persistent identifiers collected by the SDKs, we also found several SDKs are harvesting context information irrelevant to their missions: TalkingData and Tencent Analytics fetch the SSIDs of nearby Wi-Fi access points and TalkingData even collects the list of installed apps on the device. Why these information is needed is not yet clear to us but such behaviors should be prohibited, due to the sensitive nature of the data (e.g., the list of installed apps could leak user's personal interest).

Weak Protection over Network Communications. The data transfered from the user to the analytics server should be encrypted, to protect against passive MITM adversary who breaches user's privacy and active MITM adversary who manipulates usage statistics. We examined whether network communications are sufficiently protected by analytics but have only confirmed 4 SDKs deploying HTTPs to protect the confidentiality and integrity of data transition. Among the remaining, UMENG and TalkingData do not even encrypt the data. Instead, they encode the data in the HTTP POST sessions, which could be easily recovered. Tencent Analytics encrypts the data with RC4, but the key is hard coded in the app. Baidu analytics encrypts their data in AES CBC mode and the key and IV are transferred in HTTP header encrypted with a RSA public key. We have not discovered an effective way against this customized protocol but we suggest such method to be replaced by HTTPs which is already validated by the community.

4. DISSECTING FRAUD SERVICES

Inflating the usage statistics is not only feasible in theory, but also carried out by real-world fraudsters and becoming a business. We discovered two such services, named *Laicaimao* and *Anzhibao*, open to app developers publicly. They are mainly targeting markets in China and have been running for years. We infiltrated them by joining their platforms and running campaigns towards inflating the usage statistics on the analytics platforms we surveyed. We hope our study could help the community gain better insights into this new type of "gray business" and shed lights to new solutions against fraudsters.

Experiment Settings. Our main goals for this infiltration study include understanding the operational model of these services, how the back-end resources are orchestrated and how effective they are in inflating the usage statistics. We built 8 testing applications with SDK integrated for all 8 analytics based on their demos. Code that invokes logging APIs of SDKs were added to `onResume` and `onPause`. In addition, we implemented a module which can probe the running environment and collect information like device model [1] and identifiers [2]. The data is then transmitted to a server operated by us to further analyze the back-end infrastructure of fraud services. For the purpose of evaluating the efficacy of these services, we registered eight accounts on the analytics platforms and requested the fraud services to promote our apps for 7 days. The reports presented by the analytics platforms were analyzed daily. We elaborated our findings below.

Infiltration of Laicaimao. Laicaimao[3] advertises itself explicitly

[1]It can be read from `android.os.Build`.

[2]Call `android.telephony.TelephonyManager.getDeviceId()` for IMEI, `Settings.Secure.getString(android.content.ContentResolver,"android_id")` for Android ID and `android.net.wifi.WifiInfo.getMacAddress()` for MAC.

[3]http://www.laicaimao.com/

Table 4: Number of active users Laicaimao brought in per day.

Analytics	Day 1	Day 2	Day 3	Day 4	Day 5- 7
Google Analytics	10	0	0	0	0
Localytics	1	0	0	0	0
MixPanel	8	0	0	0	0
Flurry	1	0	0	0	0
UMENG	8	0	0	0	0
Tencent Analytics	8	1	0	0	0
Baidu Analytics	11	0	0	0	0
TalkingData	9	4	4	0	0

as a service to boost app usage statistics. Also interesting is that it clearly mentions it could elevate the chance of attracting investors, improve the KPI (key performance indicator) of employees working on app marketing and gain better revenues from advertisements. Surprisingly, the service is free of charge. There is only one condition for using the service: it requires the member to promote the apps from others at the same time.

After signing up, a member can upload her app into the website of Laicaimao. By running the client software (a Windows application) developed by Laicaimao, the member's app together with apps from others will be downloaded and launched in an emulated Android environment locally. Apparently, the more developers are joining the platform, the more fake users can be acquired for the app.

We are interested in how Laicaimao forges user actions. To this end, we analyzed the client software through reverse-engineering of the binary. We found the client is actually built upon *BlueStacks* [5], a very popular emulator to play Android games on desktop PC. Laicaimao leverages ADB shell [15] to issue commands to BlueStacks to open the main activity of the app. The whole process is performed automatically without user's intervention. When an app is launched, it is run for 93.2s in average and then closed. Most of the time, the app is not reopened but we found occasionally the app is started on the next day to inflate the number of active users. Interestingly, page views fraud is also committed by Laicaimao: it actually runs `monkey` through ADB shell [3] to explore app's activities. For each app downloaded and launched by Laicaimao, 3.2 page views are triggered in average.

To make the usage data more convincing to analytics platforms, the client selectively hooks the APIs that have access to identifiers and replaces them with other values. The default IMEI embedded by BlueStack is `000000000000000`, and it can be directly recognized as emulator's IMEI by online checkers. Laicaimao chooses IMEI that can indicate the host is a physical device, e.g., "Samsung Galaxy Note II". Another relevant field, IMSI (International Mobile Subscriber Identity), is manipulated in the similar manner. The build properties of the emulator are modified as well to remove any indicator suggesting the host is an emulator.

Despite that Laicaimao claims it is extremely popular (downloaded for over 10 million times), the gain through using this service is rather marginal. We queried the analytics platforms every day during our campaign. The number of active users observed per day is listed in Table 4, showing that the number of users stops to grow after three days for all platforms. We speculate the cause is that the members running Laicaimao client on a daily basis are limited, which is also confirmed after we chatted with one administrator of Laicaimao. In addition, the effectiveness of Laicaimao differs greatly against different analytics: we were only able to gain one user for Flurry and Localytics. While Laicaimao feeds distinct persistent identifiers like IMEI to each client, other identifiers (i.e., Android ID) are the same across clients. Because Flurry and Localytics rely on Android ID to distinguish users, the number of users is fixed at one. In the end, we highly suspect that Laicaimao presents the download number falsely in order to boost its credibility.

As a result of its mutual promotion model, besides our app, 10 other apps were downloaded and 8 were launched by our client (2 cannot be opened and 3 were different versions of the same app). We found these apps covered various categories like games, news, social and traveling. Most of them were unpopular (less than 25K downloads in Wandoujia), except `com.baidu.appsearch` (120M+ downloads in Wandoujia).

Regarding the back-end infrastructure of Laicaimao, we examined the data collected by the probing module of our testing app. The number of seen IPs, identifiers and device information are listed in Table 6. We first matched the IPs (8 in total) with lists of known proxies and found none of them were enlisted, suggesting Laicaimao members were not bothered to hide their locations through proxy. Though the number of IPs is small, the geo-locations were rather spread, covering 7 provinces in China.

Due to the small number of IPs identified, it is possible that Laicaimao was not active or in the maintenance stage during the period of our experiment. Hence, we relaunched our campaign six months later (in June, 2016) to validate this concern. Again, we only found 9 unique IPs (see Table 6). However, none of them overlap with the IPs we discovered previously. Laicaimao was still operated and it actually changed its strategies of faking device information: we found 3 manufacturers and 4 models were conveyed by `android.os.Build`, instead of only one detected in the previous period.

Infiltration of Anzhibao. Anzhibao[4] is a paid app promotion platform. Unlike Laicaimao, Anzhibao does not require its member to run a client software locally as a cluster of dedicated devices is set up for the fraud. The developers who want to promote their apps can just submit them through web interface, choose one customized service package, and then wait for crafted usage data being populated in analytics report. There are five options we have identified for package customization, including the number of new users recruited (denoted NS) on the first day, the ratio of active users (denoted AS) among NS after the first day (or called residency ratio), how long is the campaign (the duration for which the app is run by AS), how long is the average running time of the app, and the rate of new users recruited per day. Since the degree of engagement is vital to developers and investors, a user is priced differently on such aspect: a user only showing up on the first day costs 0.1 Chinese Yuan (cheapest option), while it cost 1.5 Chinese Yuan for a user who uses the app every day at least 5 minutes for 60 days.

To verify the efficacy of Anzhibao, we purchased two types of services from its website to promote our apps. For the first experiment (experiment I), we acquired 100 users who showed up only on the first day for each testing app, costing in total 80 Chinese Yuan or around 12 US Dollars (0.1 Chinese Yuan/user $\times$ 100 users/app $\times$ 8 apps). For the second experiment (experiment II), we acquired 100 users per day for seven days (30% of the users launch the app for more than one day) to promote the testing app integrating Tencent Analytics, costing 210 Chinese Yuan or around 32 US Dollars (0.3 Chinese Yuan/user $\times$ 100 users/day $\times$ 7 days). We examined the analytics reports provided by the eight platforms and found out Anzhibao is indeed as effective as it claims. For experiment I, the number of active users can reach 100 or even more (15 new users were rewarded as bonus when paying for 100) only

[4]http://www.anzhibao.com/

Table 5: Number of active users Anzhibao brought in per day during experiment I.

Analytics	Day 1	Day 2	Day 3	Day 4	Day 5- 7
Google Analytics	112	6	0	0	0
Localytics	106	6	0	0	0
MixPanel	112	7	0	0	0
Flurry	112	7	0	0	0
UMENG	93	5	0	0	0
Tencent Analytics	111	6	0	0	0
Baidu Analytics	109	6	0	0	0
TalkingData	112	6	0	0	0

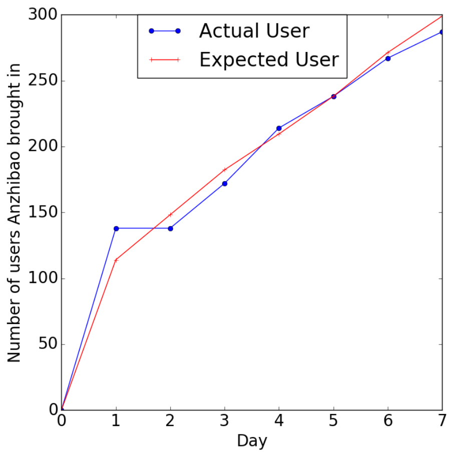

Figure 2: Number of active users Anzhibao brought in per day during experiment II.

except UMENG, as described in Table 5. For experiment II, the actual result in the report of Tencent Analytics again matched our expectations. We compute the amount of expected active users as follows: assume the number of active users is A_i and the number of new users Anzhibao brought in is B_i for the day D_i, then $A_{i+1} = B_{i+1} + \sum_{j=0}^{i} B_j * 30\%$ for $D_{i+1}(0 \leq i \leq 6)$. We compared the number of actual users and the expected users for the 7 days and the difference per day is quite small, as illustrated in Figure 2.

To maximize the usage of its service infrastructure, Anzhibao pushed multiple apps to a single device and asked for promotion at the same time. Through the probing module accompanied to our testing apps, we discovered 102 unique non-system apps from all the hosted devices, with 9 apps in average installed per device. Though suspicious, some of the apps we observed might be caught by accident: they could be pre-installed by the customized ROMs or installed by the service worker owning the device. As such, we filtered out the apps that installed by less than 10 devices, which leaves 5 apps considered very suspicious. The app with most installations (61) is `com.kufaxian.shijiazhuangshenbianshi` (5.2K+ downloads from Wandoujia [36]) and the most popular app is `com.cmbchina.ccd.pluto.cmbActivity` (5.2 M+ downloads from Wandoujia).

Similar to the study on Laicaimao, we also collected the IPs used by Anzhibao to study how the infrastructure was set up. We detected 37 IPs in total and they were spread in 12 provinces in China (see Table 6). Again, none of these IPs matched the ones used by

Table 6: Probing results on Laicaimao and Anzhibao.

Service	IP	Province	Manu[1]	Model	MAC	IMEI	AID[2]
Laicaimao[3]	8	7	1	1	16	16	1
Laicaimao[4]	9	7	3	4	9	9	1
Anzhibao[5]	37	12	19	66	117	118	115

[1] Manu: short for manufacturer
[2] AID: short for Android ID
[3] This experiment was conducted in December, 2015
[4] This experiment was conducted in June, 2016
[5] This experiment was conducted in December, 2015

open proxies. In addition, Anzhibao took more efforts in providing valid identifiers (e.g., 118 IMEIs) and options of device models (66).

One prominent feature advertised by Anzhibao is the adoption of physical devices. Such feature is attractive to customers who worry using emulators could be spotted and punished by analytics platforms. We are interested in whether this claim is true. To this end, we applied the fingerprinting techniques proposed in previous literature [25] to determine whether our app was hosted by emulator. We collected the readings from the motion sensors (accelerometers, e.g.), network settings and build models and it turns out the hosts were highly possible to be emulators, contrary to the advertisement: the readings were fixed all the time except for only one device and the build models contradicted IMEIs on the most of the devices.

Summary. Our study revealed the key strategies on how the fraud services are operated. They offer peer-to-peer or dedicated services and emulator is a favourite option to host the promoted app generated usage data. The services adopt quite simple methods to fake statistics, mostly swapping identifiers, but are still quite effective against analytics platforms. Oftentimes, multiple apps are promoted by the same device concurrently and there is a great variance among the service buyers (the developers of both popular and unpopular apps). The devices used for fraud are widely distributed, making IP-based detection likely to fail.

5. HOW USERS ARE COUNTED BY ANALYTICS SERVERS

Active users and new users are two very important results presented by analytics platforms. Our infiltration study revealed strategies adopted by the fraudsters to inflate these numbers, and the results from analytics servers reflected their efficacy. Still, our knowledge on how the servers count valid users is limited: an analytics server could use either all the collected identifiers or only one of them to label a single user; the server could scrutinize all identifiers or simply skip the checking. None of the design choices were unearthed by the infiltration study. We argue that such "reverse-engineering" process is necessary to understand the limitations of current analytics platforms, which could lead to better design choices. In this section, we try to gain the insights through "stress-testing". We built a testing platform which enumerate parameters of requests and examine the responses from servers. If the number of users reported is increased, we consider the way we mix identifiers is valid and the check on server is bypassed. We elaborate how we carried out the test and the results we obtained as follows.

Experiment Setup.

We consider all 3 persistent identifiers (IMEI, ICCID and MAC), 2 non-persistent identifiers (Android ID and GAID), 1 analytics-dedicated identifier (CID used by Google Analytics and MixPanel) as targets for manipulation. Additionally, we include IP to examine

Table 7: The number of new user counted by the analytics after we post 100 requests per identifier with different values.

Analytics	IMEI	AID[1]	MAC	ICCID	GAID	CID[2]	IP
Google Analytics	—[3]	—	—	—	—	100	1
Localytics	—	100	1	—	1	—	1
MixPanel	—	—	—	—	—	100	1
Flurry	—	100	—	—	—	—	1
UMENG	100	100	100	1	—	—	1
Tencent Analytics	100	—	100	—	—	—	1
Baidu Analytics	100	—	1	—	—	—	1
TalkingData	100	100	100	1	1	—	1

[1] AID: short for Android ID
[2] CID: Analytics-dedicated Client ID
[3] —: The identifier is not used by the analytics SDK

Table 8: How a unique user is determined by each analytics.

Analytics	User Label
Google Analytics	CID
Localytics	Android ID
MixPanel	CID
Flurry	Android ID
UMENG	MD5(IMEI) or MD5(Android ID) or MD5(MAC)
Tencent Analytics	IMEI OR MAC
Baidu Analytics	IMEI
TalkingData	MD5(IMEI+Android ID+MAC)

whether it is used to label user as well. Through studying the code of the SDKs, we found none of the requests issued from SDKs are context-sensitive. In other words, we can issue repeated requests to analytics servers without actually running the apps. Therefore, we wrote a Python script which crafts the HTTP/HTTPs requests following the formats and encoding rules specified by analytics. Each request only flips one field with an unused value and we set 10 seconds as interval between two consecutive requests. For the evaluation on IP, we tunneled our traffic to an open proxy which offers a pool of source IPs. In the end, we count the number of new users displayed by analytics platforms. We limit the number of requests to 100 per identifier/IP, to avoid causing extra caution from analytics and cost (MixPanel and Localytics ask for additional charge when the user amount is large). In total, we generated 2,700 requests to all eight analytics servers. Some servers received less requests because less identifiers are used by the corresponding SDKs.

Result Analysis. Table 7 lists the number of users counted by each analytics. First, we found not all identifiers were used to label users. For instance, ICCID were collected by UMENG and TalkingData but ignored when counting users, as all 100 distinctive ICCID were pointed to identical user. MAC and GAID were treated similarly by Localytics, UMENG and Baidu Analytics. Second, none of the platforms use IP to label users. As IP could be changed when a user moves from one place to another or switch between WIFI/GSM, it is not an ideal choice for analytics. Third, it appears that no check is performed on analytics servers to thwart our simulated attacks. Any request with a previously unseen identifier (e.g., Android ID for Localytics) leads to an increment in user number. Moreover, the basic validity check over IMEI is missed by all platforms, since all crafted IMEI were counted (we did not take any effort to make our IMEIs pass the online checker).

Based on the results above, we inferred the rules adopted by analytics in counting unique users and we present them in Table 8. We found all the rules are easy to bypass as all identifiers considered can be easily changed without much cost. To summarize, the authenticity of the analytics reports is in deed questionable under usage fraud and the investors or other audiences should be really careful when reading them.

6. FINDING IN-APP USAGE FRAUD

A developer could carry out in-app usage fraud by triggering the analytics APIs without user interactions. Qingting FM is such a case in which a hidden activity is prompted every day to inflate the number of active users, as explained in Section 2.1. We want to know whether such in-app usage fraud is an individual case or committed also by other app developers. To this end, we built a scanner based on static code analysis to capture the apps exhibiting the similar behaviors. Through reverse-engineering the code of Qingting FM, we found it registers several *broadcast receivers* to keep itself hanging in the background and running automatically without being opened by the user. It also forks *remote services*, which can be executed in their own processes at background even when the app's main process is terminated. Qingting FM leverages broadcast receivers and remote services to start the invisible activities regularly. As a result, the analytics APIs logging the page views can be invoked disregarding how the app is used.

Such observation inspired us to look for components including broadcast receivers, remote services, invisible activities and the links between them. We designed and implemented a context-insensitive static analysis tool based on SOOT[29] to detect this kind of fraud. The analysis process is similar to the method for activity transition graph construction proposed by Azim et al. [4]. Specifically, through analyzing `AndroidManifest.xml`, we obtain all the statically registered broadcast receivers and remote services (service with `android:process` set), together with all activities. We look for execution paths from broadcast receivers and remote services to invisible activities. In particular, we analyze each of their methods, search for the APIs relevant to service/activity invocation (e.g. `startService` and `startActivity`) and extract all the services and activities created by them. Next, we put the identified activities into a set called `ActivitySet` and the services into a set `ServiceSet`. For each service in `ServiceSet`, we perform the similar process to gather all the activities created by it and add them into `ActivitySet`. Finally, for each activity in `ActivitySet`, we need to decide whether it is invisible. We take two steps for this task. We first check if `theme` of the activity is set to `transparent` in the manifest file. Then, we look into the methods under each activity class and capture the one creating invisible layouts (e.g. invoking `setContentView` and `findViewById` on invisible layout elements). If all these criteria are met, we detect a valid execution path starting from a constantly running component to an invisible activity in the app, and conclude the app is engaged in usage fraud. We tested this method in a pilot experiment and found it both efficient and very accurate (the chance of false alarming is very low).

We collected 10,406 apps integrating UMENG SDK and scanned them using the approach described above. Although we found a considerable amount of apps (2,048) containing invisible activities, they are either isolated or cannot be triggered by broadcast receivers or remote services due to the absence of valid execution paths. The result suggests in-app fraud is unfavored for now, probably due to the high risk of leaving evidence on users' devices.

7. RECOMMENDATIONS

In this section, we present several recommendations for analytics platforms against the threats of usage fraud. As discussed in Section 3.2, fully mitigating the problem is very difficult, under the current settings of mobile systems. However, it is still meaningful to raise the bar against fraudsters through client-side and server-side enhancement. Our recommendations are described below:

Identity Verification. As suggested by our infiltration study, identifiers (e.g., IMEI) generated by fraudulent services were largely invalid and failed in the online check. On the other hand, none of the analytics platforms we surveyed rigorously checked the identifiers. Hence, we recommend the platforms to leverage online checkers and prune the usage logs produced from the invalid users or present them separately in the report.

Emulator Detection. Emulator is a preferred choice for fraudsters to set up the infrastructure, due to its low cost. Normally, legitimate users download and run the apps on physical devices [5]. Hence, the analytics SDK could detect the existence of emulator and report the status to the server for fraud detection. So far, the fraudsters only spent mild efforts in hiding from emulator detector, e.g., by manipulating build models. The advanced fingerprinting approach, e.g., sensor reading [25], could effectively pinpoint the existence of emulator and we recommend the analytics SDK to implement the fingerprinting capabilities. Yet, the privacy of mobile user should not be sacrificed and the result passed from the client to server should be very concise (i.e., emulator or not).

Data Analysis. Besides the efforts on client side, we argue fraud detection should also be enforced on server side through data analysis. A naive but easy to deploy approach is to blacklist the IPs of fraud devices [6]. Such IPs could be harvested through infiltrating the fraud services. A more robust approach is to capture the unusual usage patterns through machine learning. As an example, fraudsters tend to boost the usage statistics only for a limited time to prepare the usage report, therefore the analytics could look for high fluctuations of users within a short period. In addition, unsupervised learning could be applied to detect group of users exhibiting abnormal behaviors concurrently, which has been leveraged for detecting in-app purchase fraud [10, 9].

8. DISCUSSION

Ethical Concerns. We purchased packages from Anzhibao to assess the effectiveness of its service, which may raise ethical concerns as the fraudsters were funded. We want to emphasize that the infiltration study is necessary to shed light into the gray business and we have taken measures to reduce the expense to a very low level (less than 50 US Dollars), such that the profit to the fraudsters is negligible. We integrate probing modules into the testing apps to collect configuration information of clients. None of the identifiers we collected can be linked to the device owner. For the device IPs, we only queried for the associated geo-locations and checked them against blacklists without tracing back to the owners. None of the apps we developed were uploaded to app stores for public distribution and the number of recruited fake users were small, to avoid incurring manual inspection from analytics providers.

Usage Fraud in iOS Ecosystem. The same fraud is possible but the hazard is less serious in iOS ecosystem for two reasons. First, iTunes is the dominant channel of app dissemination, even in China where Google Play is blocked, as such the app ranking in iTunes is more valuable to developers and investors. So far, we have not found any rogue promotion service that aims to defraud analytics platforms on iOS. Second, building service infrastructure on top of iOS could be more expensive. There is no off-the-shelf iOS9 emulator available now and physical devices have to be purchased in lieu.

Potential Security Threats from the Fraud Services. Users of Laicaimao and Anzhibao could use the web interface to upload their apps for promotion. Though both services require that the uploaded apps should not be malicious and claim to screen the apps, we found the check is rather loose. It took only a matter of seconds for our app to pass the check, which is considerably shorter than the delay from Google Play and other third-party app stores. None of our apps were rejected, even though our apps make use of probing APIs considered harmful by Google. The infrastructure operated by the fraudsters could be turned from "gray" to "black" if the malicious apps were uploaded, e.g., botclient launching DDoS attacks.

9. RELATED WORK

Blackhat ASO (App Store Optimization). To gain an app a good ranking in app store, Blackhat ASO techniques were developed and already applied by fraudsters. Previous studies have revealed the strong correlation between app ranking and the number of app download/good reviews. Santos et al.[11] analyzed the existing Blackhat ASO techniques and their real-world impact. Xie et al. [38] studied how the fake reviews are populated on app stores. Against these fraudulent techniques, Zhu et al.[42] proposed a ranking fraud detection system by modeling apps' ranking and rating behaviors through statistical hypotheses tests. Our work focuses on a different vector of app fraud, namely the usage fraud, which aims to manipulate the statistics generated by third-party analytics providers in the purpose of cheating investors. Though still preliminary, our study has shown some interesting insights regarding this gray business and the problems underlying the analytics platforms.

Ad Fraud in Mobile Devices. A few works have studied the ad fraud in mobile apps where fetched ads are displayed or clicked without users' awareness. Liu et al.[21] proposed an automated app testing method to detect placement fraud (ads placed close to or under other UI elements) in Android apps. Crussell et al.[8] developed a system to detect fraudulent app which injects click events without user interaction or requests ads while the app is running in the background. Li et al.[20] proposed a verifiable mobile ad framework (named AdAttester) based on ARM's TrustZone technology. AdAttester collects proofs of user actions which cannot be tampered by adversarial apps and attaches them to the requests to ad providers for attestation. Yet, the techniques proposed above are ineffective against the usage fraud we studied here, in which the fraud is committed on the devices fully controlled by attackers. Attackers can disable all the security features (e.g., ARM TrustZone) required for defense and instrument system APIs to feed any value at their will.

10. CONCLUSION AND FUTURE WORKS

Despite a number of works studying fraud against users and advertisers, the usage fraud against investors is still overlooked. In this paper, we carried out the first study on this new type of threat. We reveal how fraudsters operate their business by infiltrating their services and assess the defense deployed by analytics. Our study suggests that adversary has already leveraged a variety of techniques to fool the analytics, including identifier swapping and API instrumentation. Unfortunately, usage fraud has not been consid-

[5]Some legitimate users might also run the apps on emulators, e.g., playing Android games on BlueStacks, to leverage the better I/O interface and hardware offered by PC, but the number of such users is usually small.

[6]We do not advocate blacklisting identifiers because they are easy to generate and fake, as suggested by our study.

ered as a serious threat by analytics providers as the basic checks are missing. As a result, it is trivial to add a fake user into the analytics report.

Our study is still preliminary at this stage. In the future, we plan to expand the investigation to more fraud services and draw a more comprehensive picture of the landscape of this fraud business. Meanwhile, we will also investigate feasible methods to mitigate such threat.

11. ACKNOWLEDGEMENTS

This work was supported by Tsinghua National Laboratory for Information Science and Technology (TNList) Academic Exchange Foundation, Natural Science Foundation of China (grant 61472215). We thank anonymous reviewers for their insightful comments. We also appreciate Guofei Gu and Jialong Zhang from Texas A&M University providing apps for the study of in-app usage fraud.

12. REFERENCES

[1] 360SECURITY. Qingting FM is suspected to cheat investors and advertisers, with source code analyzed (translated). http://bobao.360.cn/learning/detail/2257.html, 2015.

[2] ADLER, B. The Mobile Metrics Your Investors Actually Care About. http://info.localytics.com/blog/the-mobile-metrics-your-investors-actually-care-about, 2016.

[3] ANDROIDSTUDIO. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html, 2016.

[4] AZIM, T., AND NEAMTIU, I. Targeted and depth-first exploration for systematic testing of android apps. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications* (New York, NY, USA, 2013), OOPSLA '13, ACM, pp. 641–660.

[5] BLUESTACKS. The World's Largest Mobile Gaming Platform on PC. http://www.bluestacks.com/, 2016.

[6] CHINASKY. An encyclopedia of app promotion methods (translated). http://www.chinaskynet.net/sky-chinaf/subnews.php?ID=26, 2014.

[7] CLOVER, J. Apple Cracking Down on Fake App Store Reviews. http://www.macrumors.com/2014/06/13/apple-fake-app-store-reviews/, 2014.

[8] CRUSSELL, J., STEVENS, R., AND CHEN, H. Madfraud: Investigating ad fraud in android applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2014), MobiSys '14, ACM, pp. 123–134.

[9] DATAVISOR. Are You Paying for Fake App Installs? https://www.datavisor.com/big-data/are-you-paying-for-fake-app-installs/, 2016.

[10] DATAVISOR. Mobile Fraudsters Gone in a (Device) Flash. https://www.datavisor.com/threat-blogs/mobile-fraudsters-gone-in-a-device-flash/, 2016.

[11] DE LOS SANTOS, S., GUZMAN, A., ALONSO, C., AND GÓMEZ-RODRÍGUEZ, F. Chasing shuabang in apps stores. In *2015 APWG Symposium on Electronic Crime Research, eCrime 2015, Barcelona, Spain, May 26-29, 2015* (2015), pp. 13–21.

[12] DIAMONDBACK. Micromax Remotely Installing Unwanted Apps on Devices. http://www.xda-developers.com/micromax-remotely-installing-unwanted-apps-on-devices/, 2015.

[13] GEORGIEV, M., IYENGAR, S., JANA, S., ANUBHAI, R., BONEH, D., AND SHMATIKOV, V. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 38–49.

[14] GOOGLE. Advertising ID. https://support.google.com/googleplay/android-developer/answer/6048248, 2016.

[15] GOOGLE. Android Debug Bridge. https://developer.android.com/studio/command-line/adb.html, 2016.

[16] GOOGLE. ANDROID ID. https://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID, 2016.

[17] GRACE, M. C., ZHOU, W., JIANG, X., AND 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* (New York, NY, USA, 2012), WISEC '12, ACM, pp. 101–112.

[18] IMEIDATA.NET. iPhone ICCID Check. https://imeidata.net/iphone/iccid-check, 2016.

[19] IMEI.INFO. Check IMEI. http://www.imei.info/, 2016.

[20] LI, W., LI, H., CHEN, H., AND XIA, Y. AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2015), MobiSys '15, ACM, pp. 75–88.

[21] LIU, B., NATH, S., GOVINDAN, R., AND LIU, J. DECAF: Detecting and Characterizing Ad Fraud in Mobile Apps. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2014), NSDI'14, USENIX Association, pp. 57–70.

[22] MATHEW INGRAM. Mobile botnets are costing advertisers 1 billion in ad fraud, study shows. http://fortune.com/2015/07/23/mobile-ad-fraud/, 2015.

[23] MINAKHMETOVA, D., AND SAVCHENKO, A. Mobile marketing strategy development in china, japan and south korea: An apple app store example. *Linnaeus University Dissertations* (2015).

[24] MYTHARCHER. Logs about traffic hijacking in Kunming, Yunnan by China Telecom (translated). http://yanjunyi.com/blog/posts/yunnan-telecom-intercept-record.html, 2014.

[25] PETSAS, T., VOYATZIS, G., ATHANASOPOULOS, E., POLYCHRONAKIS, M., AND IOANNIDIS, S. Rage against the virtual machine: Hindering dynamic analysis of android malware. In *Proceedings of the Seventh European Workshop on System Security. EuroSec* (2014).

[26] PORTSWIGGER. Burp Suite. https://portswigger.net/burp/, 2016.

[27] SARAHMEI. App Store Shuabang Cost in Details: 80k Yuan for Top 3 in Paid App Chart (translated). http://c.mofang.com/guonei/122-537604-1.html, 2015.

[28] SON, S., KIM, D., AND SHMATIKOV, V. What mobile ads know about mobile users. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016* (2016).

[29] SOOT. A framework for analyzing and transforming java and android applications. https://sable.github.io/soot/, 2016.

[30] STATISTA. Global mobile os market share in sales to end users from 1st quarter 2009 to 1st quarter 2016. http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/, 2016.

[31] STATISTA. Number of available applications in the google play store from december 2009 to february 2016. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, 2016.

[32] STEVENS, R., GIBLER, C., CRUSSELL, J., ERICKSON, J., AND CHEN, H. Investigating user privacy in android ad libraries. In *IEEE CS Security and Privacy Workshops, SPW* (2012).

[33] SYLVAIN GAUCHET. LEARN HOW PEOPLE USE YOUR APP AN APP ANALYTICS TOOLS ROUND-UP. http://www.apptamin.com/blog/app-analytics-tools/, 2015.

[34] TAULLI, T. Mobile Metrics Investors Really Care About. http://www.forbes.com/sites/tomtaulli/2013/05/03/mobile-metrics-investors-really-care-about/, 2013.

[35] TECHPLUTO. How To Temporarily Change Android MAC Address Without Rooting. http://www.techpluto.com/how-to-temporarily-change-android-mac-address-without-rooting/, 2015.

[36] WANDOUJIA. The Official Site (translated). https://www.wandoujia.com/, 2016.

[37] XIAO, C. Yispecter: First ios malware that attacks non-jailbroken apple ios devices by abusing private apis. http://researchcenter.paloaltonetworks.com/2015/10/yispecter-first-ios-malware-attacks-non-jailbroken-ios-devices-by-abusing-private-apis/, 2015.

[38] XIE, Z., AND ZHU, S. Appwatcher: Unveiling the underground market of trading mobile app reviews. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (New York, NY, USA, 2015), WiSec '15, ACM, pp. 10:1–10:11.

[39] XUFANG MU. Lemeng Affiliation: The largest app and software promotion platform in China (translated). http://www.muxufang.com/a/339.html, 2015.

[40] ZHANG, H., SHE, D., AND QIAN, Z. Android root and its providers: A double-edged sword. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 1093–1104.

[41] ZHIHU. App developers, what tools are you using for counting usage statistics? (translated). https://www.zhihu.com/question/19742792, 2015.

[42] ZHU, H., XIONG, H., GE, Y., AND CHEN, E. Ranking fraud detection for mobile apps: A holistic view. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management* (New York, NY, USA, 2013), CIKM '13, ACM, pp. 619–628.