MobileAppScrutinator: A Simple yet Efficient Dynamic Analysis Approach for Detecting Privacy Leaks across Mobile OSs

Jagdish Prasad Achara
Vincent Roca and Claude Castelluccia
Inria, Grenoble, France
firstname.lastname@inria.fr

Aurélien Francillon
Eurecom, Sophia-Antipolis, France
aurelien.francillon@eurecom.fr

ABSTRACT

Smartphones, the devices we carry everywhere with us, are being heavily tracked and have undoubtedly become a major threat to our privacy. As "Tracking the trackers" has become a necessity, various static and dynamic analysis tools have been developed in the past. However, today, we still lack suitable tools to detect, measure and compare the ongoing tracking across mobile OSs. To this end, we propose *MobileAppScrutinator*, based on a *simple yet efficient* dynamic analysis approach, that works on both Android and iOS (the two most popular OSs today). To demonstrate the current trend in tracking, we select 140 most representative Apps available on both Android and iOS AppStores and test them with MobileAppScrutinator. In fact, choosing the same set of apps on both Android and iOS also enables us to *compare the ongoing tracking* on these two OSs. Finally, we also discuss the effectiveness of privacy safeguards available on Android and iOS. We show that neither Android nor iOS privacy safeguards in their present state are completely satisfying.

Keywords

Third-Party Tracking, Personally-Identifiable Information (PII) leakage, Privacy, Android, iOS, Dynamic Analysis

1. INTRODUCTION

Smartphones no longer involve only the user and the communication service (GSM/CDMA) provider. The revolutionary arrival of app store model for application distribution brings a large number of new actors. In the literature, service providers to whom the user directly interacts with are considered as first-party, the user being the second-party. However, there are many additional actors whose presence is not obvious to most users: Advertisers and Analytics (A&A) companies, application performance monitors, crash reporters, or push senders to name a few. The situation has become even more complex with the development of new advertising models for In-App advertising, leading to new actors like Mobile Ad Networks, Advertisers, Ad exchange networks for real-time bidding (RTB). Globally, these actors whose services are not directly used by the user, are called third-parties.

A user may accept to have data exchanges with a first-party depending on the service provided and the legal terms and conditions upon which they agreed. However, the data collection by third-parties without explicit user consent is something abnormal. Among all third-parties present on the smartphone, the presence of A&A companies is most dominant and privacy intrusive due to economic reasons. In order to increase their revenue, advertisers want to show the user personalized Ads. Therefore A&A companies are incited to collect as much information as possible to better profile user's interests and behavior. In order to achieve this goal, they need a way to identify the smartphone/user via an identifier that can uniquely be associated with a smartphone/user. This whole process of data collection is called **"third-party smartphone tracking"** or simply **"tracking"** and the process of showing user-specific Ads based on user profile is termed as **"targeted advertising"**.

Smartphone tracking and targeted-advertising are acceptable if the user is aware of it and if he agrees to receive targeted Ads based on his personal interests. Some users could also find the presence of third-parties on the smartphone beneficial, for instance being a counterpart for free services and applications. However problems arise when A&A companies collect Personally-Identifiable Information (PII) without users' knowledge. In fact, Ad libraries sometimes also include APIs through which an application can deliberately leak user PII [1]. This creates serious privacy risks for users if proper cautions are not taken. With the rapidly growing number of smartphones, people are increasingly exposed to such risks. Moreover, a smartphone is particularly intrusive, revealing all user movements as it is equipped with a lot of sensors, and it stores a plethora of information either generated by these sensors, by the telephony services (calls and SMS), or by the user himself (e.g., calendar events and reminders). Finally, various scandals in the past (e.g.,[2, 3]) make it difficult to trust all these actors present on smartphones.

Motivation.

As "Tracking these trackers" has become a necessity, various tools have been developed in the past. These tools are based on either static analysis or dynamic analysis or

interception of network traffic through Man-in-the-middle (MITM) approach. Even though static analysis techniques do scale well, they fail on obfuscated applications and therefore, are not suited to accurately detect and measure the ongoing tracking. Similarly, MITM based approaches have the limitation with respect to SSL traffic. Dynamic analysis techniques do exist on Android, TaintDroid [4] being the most prominent work in this field. Otherwise, on iOS, there does not exist any dynamic analysis technique capable of detecting and measuring the ongoing tracking. As we lack suitable dynamic analysis tools readily available on *both* Android and iOS, there is no measurement study in the literature which provides concrete evidences of ongoing tracking as well as the comparision across mobile OSs.

Contributions.

The contributions of this paper are threefold:

1. We introduce the *MobileAppScrutinator* platform in Section 3 to detect and measure the ongoing tracking on Android and iOS. It follows a dynamic analysis approach, and it is the first platform for iOS to detect private data leakage based on dynamic analysis.

2. We test a set of the most representative applications, available on both Android and iOS, using the MobileAppScrutinator platform. Our study considers not only PII, but also modified versions of PII (e.g., after encryption or hashing) sent over the Internet in cleartext or using SSL. Tracking modified PII is a key for reliable measurements as some identifiers (e.g., WiFi MAC address, AndroidID, IMEI) are often modified before being sent. Our findings are presented in Sections 5 and 6.

3. Finally, we discuss the effectiveness of privacy safeguards available on both Android and iOS in Section 7. We show that neither Android nor iOS privacy safeguards in their present state are completely satisfying.

2. RELATED WORK

Our work can be compared with existing works on two axes: 1) Tracking measurement technology/tools and 2) Measurement of PII leakage. Below we discuss and compare our work with some most representative works along these two axes.

2.1 Tracking measurement technology

Tools to measure the ongoing tracking might be based on either interception of generated network traffic, or static analysis of the application code, or dynamic analysis of applications.

Interception of generated network traffic.

This approach is based on snooping the network data using Man-In-The-Middle (MITM) proxy. For example, MobileScope [5], based on MITM proxy, was used in WSJ study [2] to investigate the top 100 applications on both Android and iOS. However, this technique cannot be used to intercept the SSL traffic which seriously limits the effectiveness of this approach; as we see in Sections 5 and 6 that almost half of PII leakage is through SSL. Additionally, MITM approach will not be able to detect the leakage of PII generated by the system (values not known to the user and therefore, could not be searched in the network traffic). This includes different PII, for example, unique IDs generated and shared by applications and user location. Also, MITM based approach would fail in cases where user PII is modified (e.g., hashed or encrypted) before being sent (and our experiments have shown that this is rather a common practice). Finally, being a network packet analysis approach, it is not always easy or feasible to identify the application having generated the monitored traffic, which makes the (manual) analysis rather complex. MobileAppScrutinator, in contrast, makes analysis directly at the operating system level, and thus does not suffer from such limitations.

Static analysis.

Past works (PiOS [6] on iOS, FlowDroid [7], ScanDroid [8], CHEX [9], AndroidLeaks [10], SymDroid [11], ScanDal [12] and AppIntent [13] on Android) are based on statically detecting a flow of data from a PII source to a network sink. These works can be classified in two categories: 1) static tainting-based (e.g., FlowDroid [7], ScanDroid [8], CHEX [9], AndroidLeaks [10]) and 2) symbolic execution based (e.g., SymDroid [11], ScanDal [12] and AppIntent [13]). Among the ones based on symbolic execution, SymDroid [11] designs a symbolic executor based on their simple version of Dalvik VM, i.e., micro-dalvik. Similarly, ScanDal [12] designs an intermediate language, called Dalvik Core, and collects all the program states during the execution of the program for all inputs. Considering the Android's special event-driven paradigm, AppIntent [13] proposes a more efficient event-space constraint guided symbolic execution. On iOS, PiOS was designed for binaries compiled with GCC/G++ compiler and since then, Apple switched to LLVM compiler. Therefore, PiOS needs to be adapted to support the analysis of binaries compiled with LLVM. Furthermore, PiOS is not available publicly, so one needs to build it from scratch to use it to detect and measure the ongoing tracking. In general, static analysis techniques do scale well but they lack dynamic information tracking and therefore, lead to false negatives.

Dynamic analysis.

TaintDroid [4] and PMP [14] are based on dynamic analysis on Android and iOS respectively. On Android, TaintDroid is a dynamic taint-based technique to detect and measure private data leakage. However, TaintDroid has its own limitations: 1) taint-based tracking can be easily circumvented using indirect information flows [15] 2) requires to make a trade-off between false positives and false negatives ([4] did not taint IMSI due to false positives) 3) misses native code (both for taint propagation and as a source of information). On iOS, PMP [14] is a dynamic/runtime tool that offers the functionality of choosing access to what information a user is willing to share with a particular application. As iOS's own privacy control feature provides the same functionality, PMP [14] is an enhancement in terms of the number of different types of private data considered. In fact, PMP fails to notify users if the accessed information is being sent to a remote server or not. As existing dynamic analysis tools on iOS are not sufficient to measure the ongoing tracking on iOS, one possibility could have been a taint-based dynamic analysis technique. However, this is not be possible to do on an iOS device because the code of applications is native (C, C++ and Objective-C). Prop-

agating taint would require to emulate native code, which involves serious changes to the system and would have a significant performance penalty. Therefore, we opted for a dynamic analysis approach, described in the next section, which can be used on both Android and iOS. To measure the effectiveness of MobileAppScrutinator on Android with TaintDroid, we perform tests on identical set of applications using TaintDroid and MobileAppScrutinator.

2.2 Measurement of PII leakage

To best of our knowledge, no previous work provides a complete picture of tracking on Android and iOS. Web-browser tracking has been thoroughly studied [16, 17], but it is not the case with smartphone tracking. We are first to provide detailed analysis and measurement data for both Android and iOS. [18] sheds some light on third-party tracking being taken place on Android using TaintDroid but is not as comprehensive as ours. Also, all other static and dynamic analysis tools proposed in the literature, for example, PiOS [6] and TaintDroid [4], analyzed some applications and presented a number of applications leaking user PII, but none of them presented a complete analysis as we do in Section 5 and 6. Also, as tracking technologies rapidly change with OS revisions, it is crucial to have up-to-date tools and a recent picture of tracking technology. Furthermore, we also consider the remote servers where PII is sent and attempted to distinguish them among first and third-parties, with the available information.

3. MOBILEAPPSCRUTINATOR

Design choices.

From tracking detection and measurement point of view, it is ideal to analyse what applications are doing at operating system level. However, we want to have a system that does not require too intrusive modifications of the OSs and does not have too many false positives (unlike dynamic tainting-based techniques). The system should work well with non-malicious application on real devices so that it can be used by anyone. Thus the design of MobileAppScrutinator starts with a simple approach: *intercepting the source, sink and data manipulation system APIs.* As the same approach is applied to both Android and iOS, it enables us to have a comparative view of tracking on these OSs. Even though MobileAppScrutinator is based on this simple approach, its implementation on Android and iOS *differs significantly* due to the differences of these OSs. However, the basic governing philosophy remains the same.

Overall architecture.

As developer APIs are public on both Android and iOS, we are able to identify all source and sink methods (i.e., methods related to access or modification of private data along with network operations either in clear-text or encrypted). MobileAppScrutinator hooks these APIs and includes extra code to these APIs. This added extra code collects various information from the application environment. Specifically, it collects information about PII being accessed, modified, or transmitted by an application along with the information about that application. Any access, or modification, or transmission of PII corresponds to an event and is stored locally in an SQLite database. This database is later analyzed automatically to detect and measure privacy leaks.

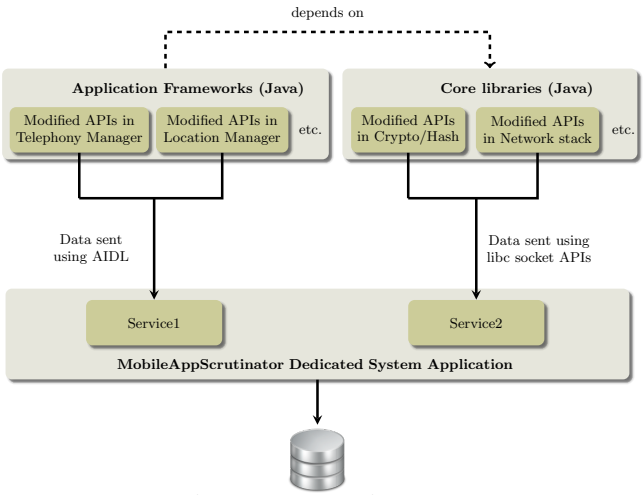


Figure 1: MobileAppScrutinator Android implementation.

In the next two subsections, we give details of how MobileAppScrutinator is implemented on Android and iOS.

3.1 Implementation on Android

As the Android source (from Android Open Source Project (AOSP)) is publicly available, MobileAppScrutinator directly modifies source code of various APIs in Java frameworks. This modified source is compiled and a new system image is generated. We develop and add a system application to this new system image. This system application runs two Android services that are responsible for receiving data from different sources. The App is also responsible for storage of data in a local SQLite database.

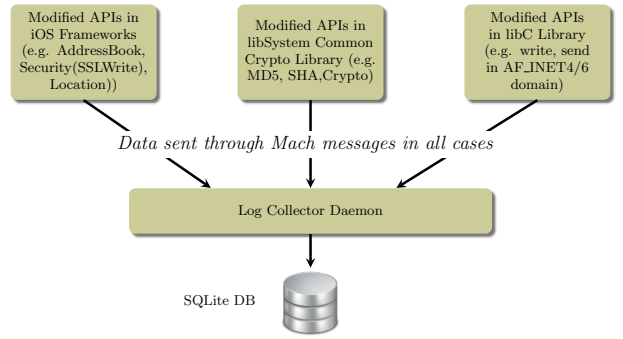
Here it is to be noted that Android application frameworks are written on top of (or utilizing) core Java frameworks, i.e., during compilation of Android source, core Java frameworks are compiled before the Android application frameworks. As AIDL[1] is part of Android application framework, it cannot be used within core Java frameworks to send data to MobileAppScrutinator system application. Thus, in modified APIs of core Java frameworks, MobileAppScrutinator uses socket APIs of libc library to send data to a dedicated service running inside MobileAppScrutinator system application. Fig. 1 provides a broad picture of how MobileAppScrutinator is implemented on Android OS.

3.2 Implementation on iOS

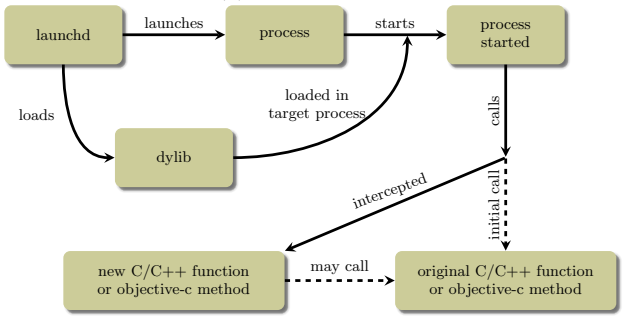
Fig. 2a depicts an overall picture of implementation of MobileAppScrutinator on iOS. The APIs of interest in iOS frameworks are modified to capture and send data related to application context as well as the PII being accessed, or modified, or transmitted (clear-text or SSL). The data communication between various processes running our custom code and the daemon is through *mach* messages. In order to execute self-signed code and get privileged access to the system, the default iOS software stack needs to be modified to remove the restrictions imposed by Apple (a technique known as "Jailbreaking" in the iOS world).

On iOS, developers may write code in C, C++ and Objective-

[1] `http://developer.android.com/guide/components/aidl.html`



(a) Overall picture.



(b) Loading *dylib* into a process.

Figure 2: MobileAppScrutinator iOS implementation.

C languages. In fact, all iOS executables are linked with the Objective-C runtime [19] and this runtime environment provides a method called *method_setImplementation*. Therefore, we use this method to change the implementation of existing Objective-C methods whereas to change the implementation of C/C++ functions, we use trampoline technique [20]. MobileSubstrate [21], an open-source framework, greatly simplifies this task. Finally, the source code responsible for modification of various APIs of interest is compiled into a dynamic library (*dylib*) which is loaded using *launchd* [22], into all or subset of running processes. Fig. 2b depicts how a *dylib* is loaded into a process using *launchd*.

3.3 Post-analysis of SQLite Data

The events stored in local SQLite database are processed by automated Python scripts. It is a two-step process: a first pass over the database on a per-application basis results into a JSON file, and a second pass over the JSON file derives various statistics.

Our first level analysis consists of the following steps:

1. Find all types of PII accessed by each application.
2. Check if PII is really sent over the network or not, and if yes, to which server it is sent to.
3. Search for the PII in the input to data modification APIs (cryptographic and hashing) and if found, look for the result in the data sent over the network.

First-level analysis results into a JSON file that stores 1) accessed PII, 2) PII passed to encryption or hash APIs 3) (un)modified PII sent over the Internet in cleartext or using SSL. Once the first pass over the database is finished, the resulted JSON file containing per-App details is processed to infer or derive various statistics. Here it is worth to mention that various PII accessed by an application are searched only in the network traffic and hash/cryptographic calls of that specific application.

In various APIs, the access to data is at byte level and in this case, the raw bytes are first attempted to be decoded using UTF-8 encoding. Since a different encoding may be used or in case of binary data, the hexadecimal representation of these raw bytes is also stored alongside. Searching in the network, or in the input to cryptographic or hash APIs is done for both UTF-8 encoded data and hexadecimal representation of the raw bytes.

3.4 Limitations

The PII leakage would remain undetected if the data is modified by the application developer using custom data modification functions before sending over the network. If the PII is modified using OS provided data modification APIs (e.g., encryption, hashing) before sending over network, MobileAppScrutinator would correctly be able to detect the PII leakage. As an App developer is not bound to use system provided hash and encryption APIs, MobileAppScrutinator might miss some PII leakage instances.

Also, Android implementation only deals with the Java code and would miss the detection of PII leakage if the direct calls are made by applications to C/C++ APIs using JNI framework. Here it is worth mentioning that it's not the limitation of the approach but rather limitation of implementation of MobileAppScrutinator on Android.

4. EXPERIMENTAL SETUP

In order to investigate the tracking mechanisms being used by third-parties, we test 140 representative (most popular Apps in each category) free Apps available both on Android and iOS. Experiments have been conducted on iOS 6.1.2 and Android 4.1.1_r6.

We manually ran applications for approximately one minute each. We could interact with some applications during this one minute duration as others required the user to log in or sign up. We did not sign up or log in as our ultimate goal was not to track the manually entered user PII but the seamless background tracking done without any user intervention/interaction. Also, we did not set out for covering all possible execution paths as third-party library code mostly starts execution when the application is first launched.

Apart from device or operating system unique identifiers and information, we also entered other information such as addressbook, calendar events, accounts etc. This enables us to know if such data is accessed and transmitted by apps.

5. CROSS-APP THIRD-PARTY TRACKING

Smartphone users mostly use dedicated Apps rather than websites for accessing services, essentially because of the relatively small screen size and the lack of mobile-optimized web pages (even if this later aspect has largely improved). Therefore tracking is no longer performed through "third-party cookies" as in web sites (that can easily be disabled by the user) but through dedicated identifiers that we now detail.

5.1 Unique identifiers from the system

First of all, let us consider the system level unique identifiers. The situation is rather different depending on the target OS.

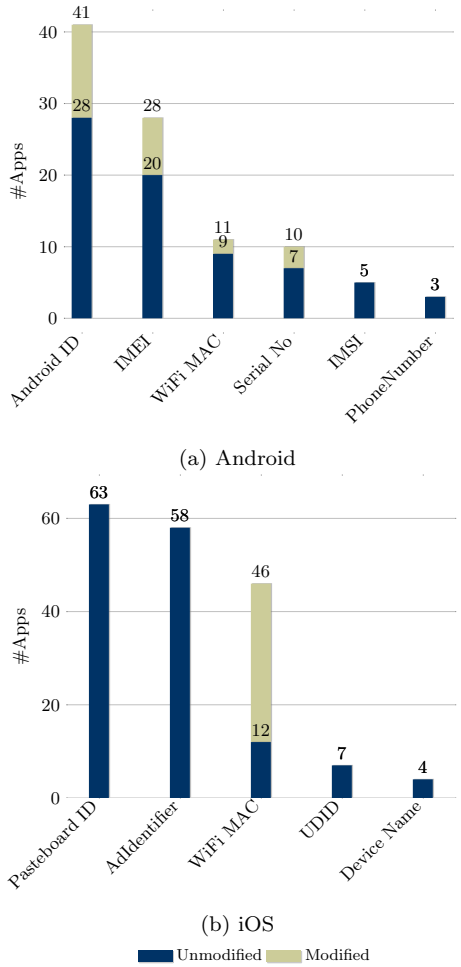


Figure 3: # Apps sending System identifiers (out of 140 Apps)

5.1.1 Android.

Various system identifiers are available on Android. A user permission is required to access hardware-tied IMEI and Wi-Fi MAC address as well as SIM-tied IMSI and phone number. OS-tied Serial Number and AndroidID identifiers are freely available to be accessed.

Our study reveals that these unique system identifiers are collected by various parties (see Table 1). Many times, both first and third-parties send these unique identifiers over the Internet to their servers unmodified (e.g., without hashing) and in clear-text (without SSL). This is a serious threat to user privacy as users can easily be identified by a network eavesdropper by simply looking at the clear-text data.

Our study reveals that third-parties collect these unique identifiers more often than first-parties, and that third-parties collect more than one identifier in most cases. Depending on the App permissions, third parties try to collect as many identifiers as they can: for instance, the *ad-x.co.uk*, *adx-tracking.com* and *mobilecore.com* third-parties collect and send the AndroidID, IMEI and WiFi-MAC address to their servers in clear-text.

Finally, we find that various unique identifiers are also sent to some IP addresses without any hostname information. It is not easy to identify to whom they belong to and why the data is sent to them. Table 1 provides the whole list of such IP addresses and the corresponding unique identifiers sent to them.

Along with these identifiers, third-parties collect and send the names of Apps in which their code is present (we notice that User-Agent http header field contains package/App name.). Knowing the Apps of a user reveals a lot about user interests [23]. The collection of this behavioral data is proportional to the number of Apps in which third-party code is present. So it is interesting to quantify the number of Apps sending these unique identifiers. Fig. 3a presents statistics about the collection of these identifiers, focusing on quantifying the presence of these third-parties in Apps. Looking at both Table 1 and Fig. 3a, it can easily be deduced that the presence of third-parties in Android Apps is huge. Globally, we find that 31% (44 out of 140) of Apps send, at least, one (un)modified unique identifier over the network.

5.1.2 iOS.

Here also our study reveals that these unique system identifiers are collected by many third-parties (see Table 2). However, there are fewer system identifiers accessible to Apps compared to Android (for instance the IMEI, IMSI and Serial Number cannot be accessed). Non surprisingly, the "AdIdentifier" is largely used, which is explicitely the reason why it has been added (Advertizing and Analytics) as a replacement to the deprecated UDID. However we notice that some companies (e.g., tapjoyads.com, greystripe.com, mdotm.com, admob.com and ad-inside.com) still did not switch to using AdIdentifer and keep on using the UDID, which has been permited for some time after its deprecation. We also observe that fewer data is sent in clear-text compared to Android.

With iOS, the device name (DeviceName) is set by the user during the initial device setup and often contains the user's real name. Since this device name is stable (the user will not generally modify it later on), even if it is not guaranteed to be unique across all devices, it is a stable identifer that can probably be used for tracking purposes. If additionally it is set with the user's real name, it also reveal its identity.

Fig. 3b presents statistics about the collection of these identifiers focusing on quantifying third-parties presence in iOS Apps. We notice that these identifiers are always collected when the user starts/stops interacting with the App. This means that third-parties can even know how long a user is using a particular App and the time when a user goes idle, revealing user habits[2].

As, globally, 60% (i.e., 84 out of 140) of Apps send, at least, one (un)modified unique identifier over the network, the risk here is huge. Comparing this number with Android reveals that third-parties presence in iOS Apps is more widespread as compared to Android but iOS Apps leak less hardware-tied system identifiers as compared to Android Apps.

5.2 Unique identifiers generated by third-parties

Let us now consider unique identifiers designed specifically by third-parties in order to bypass some restrictions of the OS. This is a practice that our study highlights on iOS

[2]A possible implication: Someone rarely using Apps after 10pm, will probably not be interested in night life (bars or clubs) and therefore Ads in this category can be concealed.

Table 1: Unique System Identifiers transmitted by 140 Android Apps tested

		Server	AndroidID		PhoneNo	IMEI		SerialNo	IMSI	WiFi MAC
			Modified	Unmodified		Modified	Unmodified			
Third-parties	Clear	amazonaws.com	✓	✓		✓				✓
		ad-x.co.uk		✓			✓			✓
		mobilecore.com		✓			✓			✓
		kochava.com		✓			✓			
		apsalar.com	✓	✓						
		mdotm.com		✓			✓			
		adtilt.com		✓			✓			
		estat.com					✓	✓		
		sophiacom.fr		✓						
		appnext.com		✓						
		flurry.com		✓						
		socialquantum.ru								✓
		sitestat.com						✓		
		pureagency.com					✓			
		smartadserver.com		✓						
		xiti.com		✓						
		playhaven.com		✓						
		yoz.io		✓						
		seattleclouds.com		✓						
		ad-market.mobi		✓						
	SSL	tapjoyads.com		✓		✓	✓	✓		✓
		airpush.com	✓	✓	✓	✓				
		revmob.com		✓			✓	✓		
		appwiz.com		✓	✓		✓			
		amazon.com		✓				✓		
		adcolony.com	✓			✓				
		fiksu.com		✓			✓			
		crittercism.com		✓			✓			
		googleapis.com			✓				✓	
		appsflyer.com					✓			✓
		dataviz.com					✓			
		mobileapptracking.com		✓						
First-parties	Clear	mobage.com		✓			✓			
		ijinshan.com					✓			
		blitzer.de					✓			
		eurosport.com							✓	
		cdiscount.com							✓	
	SSL	google.com		✓			✓	✓		
		badoo.com		✓			✓		✓	
		dropbox.com		✓				✓		
		klm.com		✓						
		airfrance.com		✓						
		airbnb.com		✓						
		groupon.com		✓						
		adobe.com							✓	
Unidentified	Clear	72.21.194.112		✓				✓		
		dxsvr.com					✓		✓	
		69.28.52.39		✓			✓			
		198.61.246.5		✓			✓			
		183.61.112.40		✓			✓			
		61.145.124.113		✓			✓			
		74.217.75.7		✓						
		183.61.112.40							✓	
		linode.com				✓				
		93.184.219.20	✓							
		107.6.111.137	✓							
		startappexchange.com		✓						
		91.103.140.6		✓						
		209.177.95.171		✓						
		ati-host.net	✓							
		adkmob.com		✓						
	SSL	fastly.net		✓						
		canal-off.sbw-paris.com					✓			

Table 2: Unique System Identifiers transmitted by 140 iOS Apps tested

		Server	AdIdentifier	UDID	DeviceName	WiFi MAC		Pasteboard IDs
						Modified	Unmodified	
Third-parties	Clear	clara.net	✓				✓	✓
		appads.com	✓					
		amazonaws.com						✓
		1e100.net						✓
		adcolony.com	✓					
		facebook.com						✓
		your-server.de						✓
		sophiacom.fr	✓					
		smartadserver.com	✓					
		mopub.com	✓					
		sofialys.net						✓
		visuamobile.com						✓
		swelen.com	✓					
		adtilt.com	✓					
		nanigans.com	✓					
		tapjoyads.com		✓				
		greystripe.com		✓				
		mdotm.com		✓				
		sofialys.net						✓
		visuamobile.com						✓
		admob.com		✓				
		ad-inside.com		✓				
		xiti.com						✓
		2o7.net						✓
		jumptap.com	✓					
	SSL	tapjoyads.com	✓				✓	
		trademob.net	✓				✓	
		adjust.io	✓					
		boxcar.io			✓			
		flurry.com				✓		
		tapjoy.com	✓					
		mobile-adbox.com	✓					
		fiksu.com	✓					✓
		tapad.com	✓					
		testflightapp.com	✓					
		adtilt.com	✓					
		nanigans.com	✓					
		ad-x.co.uk	✓					
		crittercism.com			✓			
		facebook.com	✓					
		newrelic.com						✓
		adzcore.com	✓					
First-parties	Clear	gameloft.com	✓				✓	✓
		spotify.com						✓
		disneyis.com						✓
		mobiata.com		✓				
	SSL	gameloft.com	✓				✓	
		paypal.com	✓		✓			✓
		booking.com	✓					
		eamobile.com	✓					
		expedia.com		✓				
Unidentified	Clear	amazonaws.com	✓	✓			✓	
		igstudios.in						✓
		69.71.216.204			✓			
		198.105.199.145			✓			
		akamaitechnologies.com						✓
		softlayer.com						✓
		cloud-ips.com						✓
		mydas.mobi		✓				
		mkhoj.com		✓				
		74.217.75.7/8				✓		

(that is more restrictive in terms of accessible system identifiers) but is absent from Android (that already provides all the needed system unique identifiers, some of them being even freely accessible, without having to ask for any permission). There are two motivations: (1) having an identifier independant of the system, usable even if these later become obsolete; (2) having a cross-application mechanism

As Apps are sandboxed on iOS (and Android), there must exist a way through which third-parties can somehow preserve the state across Apps to be able to track user activities across the device.

On iOS, after deprecation of UDID, third-parties had to look for other options of tracking. In fact, there exists a class called UIPasteBoard [24] on iOS that is specifically designed for cut/copy/paste operations, with the ability to share data between Apps. The data shared by Apps with this class can be persistently stored even across device restarts. Among the Apps we tested, we found that a large number of third-parties use the UIPasteboard class to share a unique third-party identifier across Apps. Out of 140 Apps we tested, 63 Apps create at least one new pasteboard entry at the initiative of a third-party library (looking at the names and types of pasteboards created and the servers where these values are sent) included in the application.

Essentially, third-party code present inside an application stores a pasteboard entry with its unique name, type and value. Later, if an App containing the code from the same third-party is installed, it retrieves the value corresponding to its pastebaord name. To have a look on pasteboard names, types and values used by various third-parties present inside 140 iOS Apps tested, please refer to Table 7 in the Appendix. Here it is to be noted that user has no control over this kind of tracking and "Limit Ad Tracking" feature of iOS is ineffective in this case.

However, on Android, we did not find Apps using this technique as various system identifiers are readily avaialble to be accessed by Apps, so third-parites do not need to generate their own identifiers. Of course, as tracking through system identifiers is stronger, there is no need to opt for this solution on Android.

6. COLLECTION OF USER PII

To create a rich user profile, third-parties can use various means to collect a wide variety of user PII:

1. By directly collecting as much information as possible from the device (i.e., by adding the appropriate code in the libraries to be included by the App developers).
2. By retrieving it from other third-parties who have already collected this information (thereby, aggregating the user PII [25]).
3. By obtaining it from first-parties.

It is difficult to measure how much information are being shared among third-parties themselves or among first and third-parties, but we can measure what kinds of data and to which extent are being collected by these third-parties directly from the smartphone. The collection of wide variety of user PII as detailed in this section shows how desperate these third-parties are to collect user PII. It also indicates the lack of legal actions as these third-parties and App developers are not afraid of legal authorities to be fined; otherwise, such tracking would have been discouraged.

6.1 PII collection on Android

Various personal data of the user is available to be accessed by Apps on Android which includes location of the user, contacts, accounts stored, etc. Our experiments with 140 Android Apps reveal that different kinds of user PII is being sent over the Internet to various first and third-parties. Table 3 presents the whole list of servers where the user PII is actually sent to.

Our study reveal that user location is sent to six third-parties in *clear-text*. In fact, it is more often sent in clear-text than using SSL to third-parties. We also find that user location is sent (encrypted or in clear-text) to nine third-parties whereas it is sent to only three first-parties. This means that user location is used more often for tracking and profiling the user and not for providing a useful service. Otherwise, we note that the name of the telephony operator and the SIM network code is being collected by a lot of first and third-parties.

Fig. 4a presents number of Apps sending different kinds of data over the Internet. We see that network code and operator name is sent by 17 and 16 Apps respectively. Moreover, as the Apps installed on a device is highly valuable information for trackers/advertisers to infer user interests and habits, we detect and measure the leakage of this information too. We find that 5 third-parties know 4 or more Apps installed by a user. Specifically, "tardemob.com", present in "Booking.com" App, collects the list of all Apps installed on the device and sends this list to its server. Interested readers may refer to Table 8 in the Appendix of the paper to know about the list of Apps and corresponding third-parties knowing them.

6.2 PII collection on iOS

iOS also makes accessible many kinds of user PII (e.g., Accounts, Location, or Contacts) to Apps. This is necessary so that a wide variety of Apps can be developed. As opposed to Android, we find that iOS Apps leak less user PII. In total, there are 8 (as opposed to 21 on Android) third-parties where user PII is sent to on iOS. Also, both first and third-parties did not send much data to their servers in clear-text.

If we look at the location data, it is sent to only two third-parties (to one in clear-text and one using SSL). There is only one third-party server and one first-party server where user location is sent in clear-text as compared to six and one respectively on Android. Globally, we note that iOS Apps sent lesser user PII over the Internet and also, they used SSL more often than their Android counterparts. Table 4 depicts details about the transmitted user PII and where this user PII is actually sent to.

Figure 4b presents the number of Apps sending user PII over the Internet. We find that 10 Apps (out of 140) send user location over the Internet as compared to 6 Apps on Android. However, more third-parties collect and send user location over the Internet on Android. This means that on iOS, the presence of a third-party is more spread over Apps even though the number of third-parties are present more on Android.

Also, we find out that iOS Apps leak more information about the list of installed Apps on the phone as compared to Android Apps. Nine third-parties know, at least, 5 names of the installed packages. Flurry, for example, knows 25 Apps installed on the phone. It is included in all these Apps and this library sends the name of App in which it is present

Table 3: User PII transmitted by a total of 140 Android Apps tested

		Server	Accounts	Contacts	Location	Operator Name	SIM Network code	WiFi Scan/Config
Third-parties	Clear	seventynine.mobi			✓	✓		
		kiip.me			✓	✓		
		google.com			✓		✓	
		3g.cn			✓			
		doubleclick.net					✓	
		goforandroid.com					✓	
		adtilt.com				✓		
		2o7.net				✓		
		nexage.com					✓	
		ad-market.mobi					✓	
		mopub.com			✓			
		mydas.mobi			✓			
		startappex-change.com					✓	
	SSL	airpush.com			✓	✓		
		appwiz.com			✓		✓	
		agoop.net			✓		✓	
		tapjoyads.com				✓	✓	
		crittercism.com				✓		
		inmobi.com						✓
		appsflyer.com				✓		
		googleapis.com	✓					
First-parties	Clear	betomorrow.com			✓	✓	✓	
		avast.com	✓			✓		
	SSL	google.com	✓	✓			✓	
		badoo.com			✓	✓	✓	✓
		checkmin.com			✓			
		groupon.de				✓		
		m6replay.fr				✓		
Unidentified	Clear	91.103.140.193	✓			✓	✓	
		adkmob.com					✓	
		dsxvr.com					✓	
		amazonaws.com				✓		
		183.61.112.40					✓	

Table 4: User PII transmitted by a total of 140 iOS Apps tested

		Server	Accounts	AddressBook	Device Name	Location	SIM Network Name	SIM Number
Third-parties	Clear	clara.net					✓	
		amazonaws.com					✓	
		bkt.mobi				✓		
	SSL	capptain.com				✓	✓	
		fring.com						✓
		crittercism.com			✓			
		boxcar.io			✓			
		testflightapp.com					✓	
First-parties	Clear	mobilevoip.com		✓				
		groupon.de					✓	
		sncf.com				✓		
	SSL	groupon.de				✓	✓	
		ebay.com						✓
		foursquare.com				✓		
		paypal.com			✓			
		twitter.com	✓					

as part of the communication with their servers. Moreover, the collection of this information is in plain-text. To get the complete list of these third-parties as well as the package names known to them, please refer to Table 9 in the appendix of the paper.

7. DISCUSSION

In order to provide transparency and control over privacy, both Android and iOS involve user decisions along with mechanisms adopted by their respective systems. However, the approach followed by Android and iOS is different: Android employs a static install-time permission system whereas iOS solicits explicit user permission at runtime. No doubt these OS mechanisms are mostly effective, they lack behavirol analysis, i.e., when, where and how often the accessed information is sent over network. For example, it is vital to distinguish the fact if the PII is sent to an application server or to a remote third-party. In fact, a user giving access to her PII for a desired service does not necessarily mean that she also wants to share this information with other parties, for example, advertisers or analytics companies. Similarly, an application accessing and sending user location only at installation time is not the same as sending it every 5 minutes.

Below we discuss the effectiveness of various privacy safeguards avaialabe on both Android and iOS based on our experiments and results.

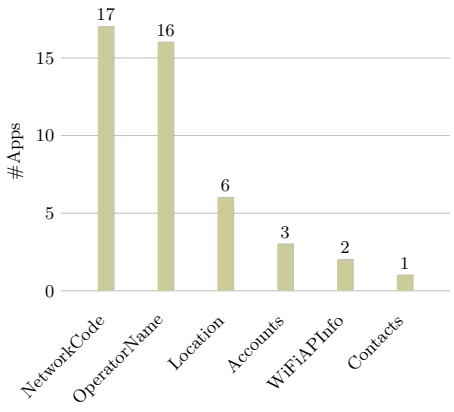
Resetting the "AdIdentifier" on iOS.

The effect of resetting the AdIdentifier is not similar to "Deleting the cookies" in web tracking and could easily be nullified. Resetting the AdIdentifier, in theory, is meant to prevent trackers from linking the user activity before and after the reset. However the trackers can easily detect the AdIdentifier change and link the two values even if Apple explicitly tells not to do so. Apps are not technically restricted by iOS to do so. In our study, we find that 20% Apps send the IdentifierForVendor[3] along with the AdIdentifier to third-parties. Table 6 shows the servers where the IdentiferForVendor is sent to. It is noticeable that many third-parties collect this identifier, whereas it was principally designed by Apple to be used only by first-parties. As IdentifierForVendor is being collected by third-parties, they are able to link the AdIdentifiers before and after the reset.

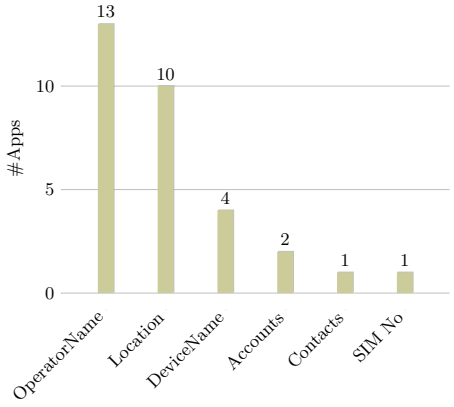
Apps bypassing the "AdIdentifier" on iOS.

We have seen that many Apps are using other tracking mechanisms to track the user in addition to the AdIdentifier. In our experiments we discovered that 93 Apps out

[3]This is a stable identifier unique to all Apps of a single developer on a particular device, that cannot be changed or reset by the user.



(a) Android



(b) iOS

Figure 4: # Apps sending PII out of a total of 140 Apps

of 140 (i.e., approx. 66%) will continue to track the user after a reset of the AdIdentifier. This measurement does not consider the applications employing the previously described technique to match the changed/reset AdIdentifiers as we cannot be sure what third-parties do with their data collected. In iOS 7, Apple banned the access to WiFi MAC Address, but the percentage only reduces from 66% to approximately 42% (60 Apps out of a total of 140 Apps.), i.e., if we exclude the Apps (24%) using only WiFi MAC address as a unique identifier for tracking.

8. CONCLUSION

This paper first introduces the MobileAppScrutinator platform for the study of third-party smartphone tracking. To the best of our knowledge, this platform is the first one that embraces both iOS and Android, using the same dynamic analysis methodology in both cases. For the first time, it provides in-depth insights on what PII is accessed, what PII is hashed and/or encrypted (possibly with other pieces of information), and what PII is sent to remote servers, either in clear-text or encrypted in SSL connections. This in-depth analysis capability is a key to analyze the applications and understand what is going on.

The second major contribution of this work is the behavioral analysis, thanks to the MobileAppScrutinator platform, of 140 free and popular Apps, selected so that they are available on both mobile OSs in order to enable comparisons. Two important aspects are considered: first we show that many stable identifiers are collected on Android, in order to track individual devices in the long term. On iOS, availability of system-level identifiers is less common, but techniques have been designed to create new cross-app, stable identifiers by third-parties themselves. The second aspect concerns the user-related information. We show that a significant amount of PII is being collected by third-parties who implicitly know a lot about the user interests (e.g., by collecting the list of Apps installed or currently running).

Finally, this work enables to have a comparative view of ongoing tracking on Andorid and iOS. Our experiments show that Android apps are more privacy-invasive as compared to iOS Apps as the presence of third-parties is cleary more in Android applications. In all cases, protective measures are required to be taken by device manufactureres, OS designers and various regulatory authorities in a coordinated way to control the collection and usage of PII.

9. REFERENCES

[1] T. Book and D. S. Wallach, "A case of collusion: A study of the interface between ad libraries and their apps," *CoRR*, vol. abs/1307.6082, 2013.

[2] "WSJ: What They Know - Mobile," http://blogs.wsj.com/wtk-mobile/.

[3] "Twitter and Path uploading user's contacts information to their servers," http://www.theregister.co.uk/2012/02/15/twitter_stores_address_books/.

[4] W. Enck and other, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones." in *OSDI'10*.

[5] "Mobilescope: Acquired by evidon," http://www.evidon.com/mobilescope.

[6] M. Egele and other, "Pios: Detecting privacy leaks in ios applications." in *NDSS*, 2011.

[7] S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 35th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2014)*, 2014.

[8] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of android applications," *Manuscript, Univ. of Maryland*, 2009.

[9] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[10] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale," in *Trust and Trustworthy Computing*. Springer, 2012.

[11] J. Jeon, K. K. Micinski, and J. S. Foster, "Symdroid: Symbolic execution for dalvik bytecode," 2012.

[12] J. Kim, Y. Yoon, K. Yi, J. Shin, and S. Center, "Scandal: Static analyzer for detecting privacy leaks in android applications," *MoST*, 2012.

[13] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*.

[14] Y. Agarwal and other, "Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, 2013.

[15] G. Sarwar and other, "On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices," in *10th International Conference on Security and Cryptography (SECRYPT)*, 2013.

[16] J. R. Mayer and other, "Third-party web tracking: Policy and technology," ser. SP '12, 2012.

[17] F. Roesner and other, "Detecting and defending against third-party tracking on the web," ser. NSDI'12.

[18] S. Han *et al.*, "A study of third-party tracking by mobile apps in the wild," Tech. Rep., 2011.

[19] "Objective-c runtime environment," http://goo.gl/OQAMqh.

[20] "Trampoline technique," http://en.wikipedia.org/wiki/Trampoline_(computing).

[21] "Mobilesubstrate framework," http://iphonedevwiki.net/index.php/MobileSubstrate.

[22] "launchd apple documentation," http://goo.gl/Ysc7Ek.

[23] S. Seneviratne, A. Seneviratne, P. Mohapatra, and A. Mahanti, "Predicting user traits from a snapshot of apps installed on a smartphone," *SIGMOBILE Mob. Comput. Commun. Rev.*, Jun. 2014.

[24] "UIPasteboard Class Reference," http://goo.gl/h1o1qX.

[25] "Privacy tools: Opting out from data brokers," http://juliaangwin.com/privacy-tools-opting-out-from-data-brokers/.

APPENDIX

Table 5: Detection of PII leakage using TaintDroid 4.3 (Same applications are tested to compare the results with MobileAppScrutinator platform).

		Server	IMEI	Browser History	Address Book	SMS
Third-parties	Clear	hinet.net	✓			
		enovance.net	✓			
		aol.com	✓			
		kimsufi.com	✓			
		typhone.net	✓			
		betacie.net	✓			
		1e100.net	✓			
		dolphin-server.co.uk	✓			
		linode.com	✓			
		amazon.com	✓			
		ati-host.net	✓			
	SSL	amazonaws.com	✓		✓	
		1e100.net			✓	
		skyhookwireless.com	✓			
		akamaitechnologies.com	✓			
First-parties	Clear	teamviewer.com	✓			
		badoo.com	✓			
		shazamteam.net	✓			
	SSL	svcs.paypal.com	✓			
		amazon.com	✓			
Unidentified	Clear	162.13.174.5	✓			
		69.28.52.38	✓			
		195.154.141.2	✓			
		188.165.90.225	✓			
		91.103.140.225	✓			
		61.145.124.113	✓			
		69.28.52.36				✓
		183.61.112.40				✓
		91.213.146.11	✓			
	SSL	31.222.69.213	✓			
		212.31.79.7	✓			
		92.52.84.202	✓			
		69.194.39.80	✓			
		72.26.211.237	✓			
		192.225.158.1	✓			
		54.256.81.235	✓			
		67.222.111.117		✓		

Table 6: Servers where IdentifierForVendor is communicated in 140 iOS Apps tested

Third-parties		First-parties	
Clear	SSL	Clear	SSL
mobileroadie.com, clara.net, appads.com, adcolony.com, sophiacom.fr, 7mobile7.com, sitestat.com, mediatemple.net	tapjoyads.com, tapjoy.com, adzcore.com, fiksu.com, crittercism.com, ad-x.co.uk	eurosport.com, gameloft.com	eamobile.com, dailymotion.com, foursquare.com, google.com, googleapis.com, paypal.com

Table 7: Different Pasteboard Names, Types and Values created by 140 iOS Apps tested

Pasteboard Names	Pasteboard Types	Pasteboard Values
fb_app_attribution, org.OpenUDID.slot.0 to 99, com.hasoffers.matsdkref, com.ad4screen.bma4s.dLOG, com.ad4screen.bma4savedata124780, com.flurry.pasteboard, com.fiksu.288429040-store, com.fiksu.pb-0 to 19, org.secureudid-0 to 99, com.ebay.identity, com.paypal.dyson.linker_id, AmazonAdDebugSettings, CWorks.5cb7c5449e677be888147c58, amobeePasteboard, com.google.maps, com.google.plus.com.deezer.Deezer, com.bmw.a4a.switcher.featureInfos and many more	com.crittercism.uuid, org.OpenUDID,public.utf8-plain-text, com.fiksu.id, public.secureudid, com.google.maps.SSUC, com.flurry.UID, com.bmw.a4a.switcher.featureinfo container,	WiFi MAC Address, 2501110D-69B7-415A-896B-4F7A83591263, ID521411E3-D88E-426E-9B7D-1060C0772C89969DC466, 3630464143444131304332308211d087-ca5b-42c3-a1a2-7b3779f6c206, 81C65A17-9F0E-4BFE-83A7-1C2C070C3353, E6644EEB-04B3-4AEF-8562-A2C29E323CCE, 55b0a791-517e-4bd4-8398-414dd527417b, And other binary data instances

Table 8: List of third-parties knowing names of installed packages on Android (out of a total of 140 Apps tested)

Third-party (Comm type)	Process Names
trademob.com(SSL)	All the processes running on the phone
google.com(SSL)	All the processes running on the phone
google-analytics.com(SSL)	com.anydo, com.rechild.advancedtaskkiller, com.spotify.mobile.android.ui, com.google.android.googlequicksearchbox, com.dailymotion.dailymotion, com.aa.android, com.comuto, com.airbnb.android
doubleclick.net(plain-text)	com.tagdroid.android, com.rechild.advancedtaskkiller, bbc.mobile.news.ww, ua.in.android_wallpapers.spring_nature
crashlytics.com(SSL)	com.evernote, com.path, com.lslk.sleepbot, com.twitter.android, com.dailymotion.dailymotion

Table 9: List of third-parties knowing names of installed packages on iOS (out of a total of 140 Apps tested)

Third-party (Comm type)	Process Names
flurry.com(plain-text)	TopEleven, Bible, RATP, Transilien, TripIt, DespicableMe, FlyAirIndia, Viadeo, Bankin', VDM, OCB, DuplexA86, SleepBot, Snapchat, Appygraph, Booking.com, foodspotting, Badoo, EDF-Releve, WorldCup2011, Quora, UrbanDictionary, babbelSpanish, MyLittleParis, Volkswagen
google-analytics.com(SSL)	InstantBeautyProduction, Evernote, LILIGO, Transilien, Viadeo, VDM, comuto, easyjet, VintedFR, Volkswagen
crashlytics.com(SSL)	dailymotion, TopEleven, AmazonFR, Path, RunKeeper, foodspotting, babbelSpanish, Deezer
urbanairship.com(SSL)	Wimbledon, RATP, HootSuite, DuplexA86, Appygraph, foodspotting, Volkswagen
xiti.com(plain-text)	laposte, ARTE, myTF1, lequipe, SoundCloud, 20minv3, Leboncoin
admob.com(plain-text)	VSC, BBCNews, WorldCup2011, RF12, UrbanDictionary
capptain.com(plain-text)	Viadeo, myTF1, rtl-fr-radios, 20minv3, iDTGV
tapjoy.com(SSL)	TopEleven, Bible, DespicableMe, OCB, MCT