Detecting Privacy Leaks in the RATP App: how we proceeded and what we found

Jagdish Prasad Achara, James-Douglass Lefruit,
Vincent Roca, and Claude Castelluccia

Privatics team, Inria, France
jagdish.achara@inria.fr,james-douglass.lefruit@inria.fr,
vincent.roca@inria.fr,claude.castelluccia@inria.fr

Abstract. We analyzed the RATP App, both Android and iOS versions, using our instrumented versions of these mobile OSs. Our analysis reveals that both versions of this App leak private data to third-party servers, which is in total contradiction to the In-App privacy policy. The iOS version of this App doesn't even respect Apple guidelines on cross-App user tracking for advertising purposes and employs various other cross-App tracking mechanisms that are not supposed to be used by Apps. Even if this work is illustrated with a single App, we describe an approach that is generic and can be used to detect privacy leaks from other Apps. In addition, our findings are representative of a trend in Advertising and Analytics (A&A) libraries that try to collect as much information as possible regarding the smartphone and its user to have a better profile of the user's interests and behaviors. In fact, in case of iOS, these libraries even generate their own persistent identifiers and share it with other Apps through covert channels to better track the user, and this happens even if the user has opted-out of device tracking for advertising purposes. Above all, this happens without the user knowledge, and sometimes even without the App developer's knowledge who might naively include these libraries during the App development. Therefore this article raises many questions concerning both the bad practices employed in the world of smartphones and the limitations of the privacy control features proposed by Android/iOS Mobile OSs.

Keywords: Android, iOS, Privacy, RATP App

1 Introduction

In the age of information technology, the ways through which user's privacy can be invaded has outgrown, and today, it has become even worse with the ubiquitous use of smartphones. So it is very critical to analyze the privacy risks caused by the use of smartphones and the mobile Apps. However, analyzing Apps to detect private data leakage is not a trivial task considering: 1) the closed-nature of some of the smartphone OSs, and 2) the need to reverse engineer sophisticated techniques employed by Mobile OSs. Among all the mobile OSs available

today, we target Android and iOS because they cover more than 90% of the whole smartphone OS market share [20] and represent two different paradigms of mobile OSs (closed-source nature of iOS while Android being open-source to some extent).

We analyze iOS and Android Apps by using a combination of static and dynamic analysis techniques, taking advantage of our instrumented versions of the OSs. We illustrate this methodology and detail our findings with the RATP App. The RATP is the French public company that is managing the Paris subway (metro). It provides a very useful smartphone App that helps users to easily navigate in the city. We show that the current iOS/Android versions (at the time of writing) of this App leak many private data to third-party servers, which totally contradicts the In-App privacy policy.

Beyond this discovery we discuss the situation and the trend we observe in terms of smartphone users tracking with stable identifiers, and some of the non-trivial techniques being used to collect information on these smartphones. All this happens without the user knowledge, and sometimes even without the App developer's knowledge who might naively include several A&A libraries in the App. We discuss the current situation and raise some questions regarding the bad practices employed in the world of smartphones as well as the responsibilities of Apple and Google. The privacy control features that are provided by these Mobile OSs are, in our opinion, both too limited and almost systematically bypassed by the A&A libraries. Apple and Google cannot ignore this situation.

The paper is structured as follows: we detail the analysis of the iOS version of the RATP App in section 2 and section 3 does the same for the Android version. Section 4 presents some related work and finally we conclude with a discussion of the responsibilities of the various actors.

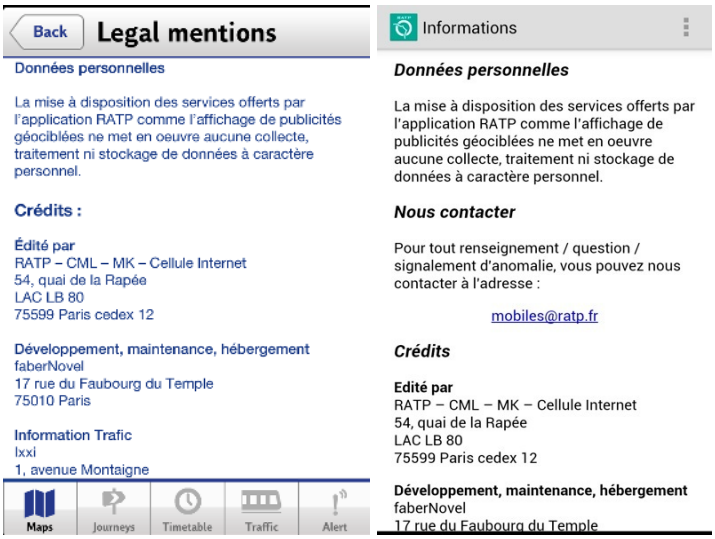
2 The RATP iOS App

2.1 Instrumented version of iOS

In order to detect if an App accesses, modifies (e.g. hashes or encrypts), or leaks some private data over the network, we first instrumented the iOS Mobile OS. Since most of the iOS Apps are written in C/C++/Objective-C, this is done by loading our custom dynamic library (dylib) at the App process-launch time: Objective-C run-time provides a method to change the implementation of the existing methods at run-time, and in case of C/C++ functions, this is done at assembly language level. In practice, we use the MobileSubstrate [17] framework that simplifies this task by providing higher-level API for replacement of C/C++ functions and Objective-C methods. So our custom library changes the implementation of some well-chosen Objective-C methods and C/C++ functions in order to catch interesting events and stores these events, with the associated parameters and/or return values, in a local database for later analysis.

2.2 Privacy leaks to the Adgoji company

The privacy policy (in French; See Figure 1) of RATP's iOS App (version 5.4.1) claims: *"The services provided by the RATP application, like displaying geo-targeted ads, does not involve any collection, processing or storage of personal data"* (translated from the French version).



(a) iOS version (b) Android version

Fig. 1. In-App privacy policies of the iOS and Android RATP Apps.

However, in total contradiction to the privacy policy, the RATP App sends over the network the MAC Address of iPhone's WiFi chip, the iPhone's name, and the list of processes running on it (which reveals a subset of the Apps installed on your smartphone) among other things, to a remote third-party. The Listings 1.1 and 1.2 show the data we captured on our iPhone while being sent over the network by the RATP App. One good news, though: this data is sent through SSL, not in clear, which avoids eavesdropping.

Fortunately, it is not trivial to detect all the Apps installed on the iPhone:

- iOS doesn't provide an API to do so, and
- due to sandbox restrictions [15], an App cannot peek into other system activities or Apps.

However, since this is a highly valuable information to infer the user's interests, techniques exist to identify some of them. Here are two techniques, both of them being used by the RATP App:

Listing 1.1. Data sent through SSL by iOS App of RATP (Instance 1)

```
UTF8StringOfDataSentThroughSSL = {"p":["kernel_task","launchd",
    "UserEventAgent","sbsettingsd","wifid","powerd","lockdownd",
    "mediaserverd","mDNSResponder","locationd","imagent","iaptransportd",
    "fseventsd","fairplayd.N94","configd","kbd","CommCenter","BTServer",
    "notifyd","aggregated","networkd","itunesstored","apsd","MyWiCore",
    "distnoted","tccd","filecoordination","installd","absinthed","timed",
    "geod","networkd_privile","lsd","xpcd","accountsd","notification_pro",
    "coresymbolicatio","assetsd","AppleIDAuthAgent","dataaccessd",
    "SCHelper","backboardd","ptpd","syslogd","dbstorage","SpringBoard",
    "Facebook","iFile_","Messenger","MobilePhone","MobileVOIP",
    "MobileSafari","webbookmarksd","eapolclient","mobile_installat",
    "AppStore","syncdefaultsd","sociald","sandboxd","RATP","pasteboardd"],
    "additional":{"device_language":"en","country_code":"FR",
    "adgoji_sdk_version":"v2.0.2","device_system_name":"iPhone
    OS","device_jailbroken":true,"bundle_version":"5.4.1",
    "vendorid":"CECC8023-98A2-4005-A1FB-96E3C3DA1E79","allows_voip":false,
    "device_model":"iPhone","macaddress":"60facda10c20", "asid":
    "496EA6D1-5753-40B2-A5C9-5841738374A2","bundle_identifier":
    "com.ratp.ratp","system_os_version_name":"iPhone OS","device_name":
    "Jagdish's iPhone","bundle_executable":"RATP",
    "device_localized_model":"iPhone","openudid":
    "9c7a916a1703745ded05debc8c3e97bedbc0bcdd"}, "e":
    {"782EAF8A-FF82-48EF-B619-211A5CF1F654":[{"n":"start",
    "t":1369926018,"nonce":"IEx9HAzG"}]}}
```

Listing 1.2. Data sent through SSL by iOS App of RATP (Instance 2)

```
UTF8StringOfDataSentThroughSSL = {"s":["fb210831918949520",
    "fb108880882526064","evernote","fbauth2","fbauth","fb","fblogin",
    "fspot-image","fb308918024569","fspot","fsq+
    pjq45qactoijhuqf5l21d5tyur0zosvwmfadyw0pvd4b434e+authorize",
    "fsq+pjq45qactoijhuqf5l21d5tyur0zosvwmfadyw0pvd4b434e+reply",
    "fsq+pjq45qactoijhuqf5l21d5tyur0zosvwmfadyw0pvd4b434e+post",
    "foursquareplugins","foursquare","fb86734274142","fb124024574287414",
    "instagram","fsq+kylm3gjcbtswk4rambrt4uyzq1dqcoc0n2hyjgcvbcbe54rj+post",
    "fb-messenger","fb237759909591655","RunKeeperPro","fb62572192129",
    "fb76446685859","fb142349171124","soundcloud","fb19507961798",
    "x-soundcloud","fb110144382383802","mailto","spotify","fb134519659678",
    "fb174829003346","fb109306535771","tjc459035295","twitter",
    "com.twitter.twitter-iphone","com.twitter.twitter-iphone+1.0.0",
    "tweetie","com.atebits.Tweetie2","com.atebits.Tweetie2+2.0.0",
    "com.atebits.Tweetie2+2.1.0","com.atebits.Tweetie2+2.1.1",
    "com.atebits.Tweetie2+3.0.0","FTP","PPClient","fb184136951108"]}
```

1. Listing the running processes using `sysctl` [4]: We decrypted (see [1] for indications of how to proceed) the RATP binary and then, opened it in a hexeditor; we searched for `sysctl` and found it (see Figure 2). This confirms the use of `sysctl` in the RATP App code (i.e. written by the developer, not coming from system frameworks/libraries), and this is the method used to get the process list of Listing 1.1;
2. Detecting if a custom URL can be handled or not: It is also possible to use the *canOpenURL* [5] function of *UIApplication* class (see [6] to know more about URLSchemes). If the URL is handled, the presence of a particular App is confirmed, otherwise the App is not installed. A major drawback of this technique is of course the need to do an active search for each targeted App, but otherwise it is a very efficient technique. The RATP App uses this technique too, as shown in Listing 1.2 which lists the URLs handled by the iPhone, thereby confirming the presence of the corresponding Apps.

Once collected, data is sent to the *sdk1.adgoji.com* (*175.135.20.107*) server owned by Adgoji [2], a mobile audience targeting company. This is again confirmed by a static analysis of the App. Figure 3 is a screenshot showing the decrypted RATP app binary opened in IDA Pro [12]. We see some methods inside the *AppDetectionController* and *AdGoJiModel* classes. Figure 4 shows the name of header files generated by running class-dump-z [9] on the decrypted binary revealing the classes from Adgoji company starting with prefix Adgoji. So the internals of the RATP App reveal that it uses the Adgoji library which, in turn, does the job of collecting and sending the information to their server.

To summarize, Adgoji tries to detect the Apps present on the smartphone in order to profile the user based on its interests, Adgoji collects the MAC Address, a permanent unique identifier attached to the device, in order to keep the device, as well as the OpenUDID, a replacement to the now banned UDID and which is used as a permanent identifier too. They also collect the name of iPhone, either to know more on the user (this name is often initialized with the real user's name, in our case"Jagdishs iPhone"), or to use it as a relatively stable identifier since the probability the device name changes over the time is very low. Finally they collect the Advertising Identifier (the `"asid"` entry), which is an acceptable practice as it is under the control of the user. How does the Adgoji company process and/or store this data? Does it further share it with other companies? We cannot say. The question remains open and only the parties involved can answer.

2.3 Privacy leaks to the Sofialys company

In addition to Adgoji, the RATP iOS App also sends the data mentioned in Listing 1.3 to the *88.190.216.131* IP address. This IP address belongs to Sofialys [21][1], another mobile advertising company. However, this time data is sent **in cleartext** which is not acceptable: we don't see any point not to use secure

[1] Whois [24] and other web services (like infosniper [13] or DShield [10]) reveal *so-par-onl-vip01.sofialys.net* as the hostname of the machine. Second level domain *sofialys*

Listing 1.3. Data sent by iOS App of RATP in cleartext

```
UTFStringOfDataSentInCLEAR = {"uage":"","confirm":"1", "imei":
    "9c7a916a1703745ded05debc8c3e97bedbc0bcdd",
    "osversion":"iPhone6.1.2",
    "odin":"1b84e4efaf650cb9a264a2ff23ca7a67b9bd72f6","umail":"",
    "carrier":"", "user_position": "45.218156;5.807636",
    "long":"",
    "ua":"Mozilla/5.0(iPhone;CPUiPhoneOS6_1_2likeMacOSX)AppleWebKit
    /536.26(KHTML,likeGecko)Mobile/10B146","footprint":{"v1":
    {"i":"3739335834508445""b""c5kkekILx11ghUfu3Ht43bUZWcHHBNbR0
    9AO4it+wtPPCBJagCIo7tgBdMlq6T244EwHnKRzeh1ybrMhKy2SztEU5tD5u5Q
    7HAisR57BYIun9aQdp0NsXwp7BXhohS92daScYcMDALqKQhYKZDriEjqW
    wtjvR9MrIKfE52EwNcA9CJJkUIT9q7sXkqkvaloOM7tMrNdMiIQYyHOtdNJ+
    ax7Ujau/IQ4pPasSXk/m6BIFsAFhjFOng0NuSwtL7e7r95s8wQhWy+
    EvJUChPIvIRXZYldCbjfdkrkvNgHZcH59Fj0dBz9Ugbyoj4a/Z60SlU+
    EatvNswORMQqdE8djVJmXkGCmwoheU10uQatr4pqA="}},"ugender":"",
    "os":"iPhone", "adid": "496EA6D1-5753-40B2-A5C9-5841738374A2",
    "uphone":"","sdkversion\":"5.0.3","test":"","lat":"","udob":"",
    "pid":"4ed37f3f20b4f","lang":"fr_FR","network":"wifi",
    "time":"2013-05-3015:45:04","alid":"186","sal":"","uzip":""}
```

connections. More precisely, the App sends the UDID (Unique Device Identifier) of the iPhone (erroneously called IMEI in the captured screenshot), as well as the precise geolocation of the user (one can enter the longitude/latitude mentioned and he'll find where we are working with a 20 meters precision), and the Advertising Identifier (which is acceptable, as explained above). Except the Advertising Identifier, everything else happens without the user knowledge.

2.4 What about Apple's responsibility?

Apple gives users the feeling that they can control what private information is accessible to Apps in iOS 6. That's true in case of Location, Contacts, Calendar, Reminders, Photos, and even your Twitter or Facebook accounts but Apps can still access other kinds of private data (for example, the MAC Address, Device Name, List of processes currently running etc.) without users' knowledge. To avoid device tracking, Apple has deprecated the use of UDID and replaced it by a dedicated 'AdvertisingID' that a user can reset at any time. This is certainly a good step to give control back to the user: by resetting this advertising identifier, trackers should not be able to link what a user has done in the past with what (s)he would be doing from that point onward with respect to his online activities. But apparently, the reality is totally different: **the Advertising Identifier only gives an illusion to the user that he is able to opt-out from device tracking because many tracking libraries are using other stable identifiers to track users.** Below are few techniques that Apps are already using or might use them:

points to Sofialys as the company this machine might belong to. And finally, the icon [22] almost confirms this as it is the same as on Sofialys web page [21].

1. Apps can access the WiFi MAC address (again through `sysctl` function in libC dylib) to get a unique identifier permanently tied to a device which cannot be changed. Fortunately, the access to the MAC address seems to be banned from the new iOS 7 version [7];
2. Apps can use UIPasteboard [23] to share data (e.g. a unique identifier) between Apps on a particular device. For example, the Flurry analytics [11] library, also included in this App binary, is doing it! Flurry creates a new pasteboard item with name *com.flurry.pasteboard* of pasteboard type *com.flurry.UID* and stores a unique identifier whose hexadecimal representation is: <49443337 38383436 44452d32 3138302d 34414231 2d423536 432d3936 38363839 36443736 35333532 30443544 3338>. Many other analytic companies (Adgoji for instance) use the OpenUDID [18], which is based on the use of UIPasteboard to share data between each other. Resetting the advertising ID will not impact these IDs;
3. Apps can use the device name as an identifier, even if it is far from being a unique identifier. People generally don't change it periodically. Even if it is not unique, this is in practice a relatively stable identifier;
4. Apps can simply store the advertising identifier in some permanent place (e.g. persistent storage on the file system), and later, if this ID has been reset by the user, they can link it with the new advertising identifier. That's so trivial to do.

We see that there are several effective techniques to identify a terminal in the long term, and **Apple cannot ignore this trend**. Apple needs to take some rigorous steps in regulating these practices.

Also, we don't understand why Apple is not giving control to the user to let him choose if an App can access the device name or not (we've seen that this is an information commonly collected by Apps). We have developed an extension to the iOS 6 privacy dashboard to demonstrate its feasibility and usefulness. Our extension to the privacy dashboard lets users choose if an App can access the device name and if it can access the Internet (See our privacy extension package [14]). It is surely not sufficient, but it is required.

The Apple privacy dashboard added in iOS6 does not help so much:

- A&A libraries included by the App developer have access to the same set of user's private data as the App itself. However, a user granting access to his/her Contacts to an App does not indicate consent for this data to be shared with other third-parties (in particular A&A companies). Whether and where the personal information is sent, is not under the control of the user via the privacy dashboard;
- We believe that an authorization system that does not consider any behavioral analysis is not sufficient. For instance, accessing the device location upon App installation to enable a per-country personalizing is not comparable to accessing the location every five minutes. That's a fundamental limit of the privacy dashboard system (and the Android authorization system too);
- Also, the permissions for accessing certain private data require a finer granularity. For instance, accessing the city/state level location or the exact lon-

Listing 1.4. Data sent in cleartext by Android App of RATP

```
DataSentInCLEAR =
    {"user_position": "45.2115529;5.8037135","ugender":"",
    "test":"","uage":"0","imei": "56b4153b8bd2f6fd242d84b3f63e287","napp":
    null,"uemail":"","pid":"4ed37f3f20b4f","alid":"114","uzip":"",
    "osversion":"3.0.31-g396c4dfdirty","lang":"en_En","sal":"","network":
    "na","adpos":null, "time":"Tue Jun 04 12:05:39 UTC+02:00 2013",
    "sdkversion":"3.2", "ua":"Mozilla\/5.0(Linux; U; Android 4.1.1;
    fr-fr; Full AOSP on Maguro Build\/JRO03R) AppleWebKit\/534.30
    (KHTML, like Gecko) Version\/4.0 Mobile Safari\/534.30","udob":"",
    "carrier": "Orange F","longitude":"0.0","latitude":"0.0",
    "freespace":null,"unick":null}]
```

gitude/latitude should be considered differently: certain Apps do not need the exact location of the user to provide the desired functionality and a user should not have to grant access to his/her precise location. The same is true for Address-book and other kinds of private data.

3 The RATP Android App

3.1 Instrumented version of Android

We also analyzed the Android version of the RATP App (version 2.8). Here we use Taintdroid [27] and in addition we changed the source code of Android itself when required. We only changed the APIs of interest, like the network APIs to look for the private data sent over the network. In addition, we use static analysis of the App to confirm some observations.

3.2 Privacy leaks to the Sofialys company

Figure 1 shows the same privacy policy as that of the iOS version. However personal information is still collected and transmitted. Listing 1.4 shows data captured while it is being sent to the network **in cleartext** by the RATP's Android App.

The above data contains some very sensitive information about the user, in particular:

- The exact location of the user: however the precision is lower than with the iOS App (a few hundred meters);
- the MD5 hash of the device IMEI: sending a hashed version of this permanent identifier is better than nothing. However, getting back to the IMEI from its hash is feasible, and even easy given some information about the device. For example, if the smartphone manufacturer and model are known, it only takes less than 1 second on a regular PC (See Figure 5) to recover the IMEI.
- The SIM card's carrier/operator name.

Listing 1.5. Permissions required by Android App of RATP

```
com.fabernovel.ratp.permission.C2D_MESSAGE;
com.google.android.c2dm.permission.RECEIVE;
android.permission.READ_PHONE_STATE;
android.permission.VIBRATE;
android.permission.INTERNET;
android.permission.ACCESS_FINE_LOCATION;
android.permission.ACCESS_COARSE_LOCATION;
android.permission.READ_CONTACTS;
android.permission.ACCESS_NETWORK_STATE;
android.permission.READ_CALL_LOG
```

Here also, the data is sent to a Sofialys server at IP address *88.190.216.131* IP address. This is confirmed by static analysis of the App: we de-compiled the Apps dex file executed by Dalvik Virtual Machine. We found two third-party packages: Adbox (with package name *com.adbox*) and HockeyApp (with package name *net.hockeyapp*) (See Figure 6 listing class descriptors). It confirms that the Sofialys Adbox library is included in the Android version (just like the iOS version). It is disturbing to see that the RATP continues with these bad practices whereas the private information leakage to *Sofialys* has already been highlighted in the past [8].

Let us have a look at the permissions the App asks (Listing 1.5). The RATP's Android App asks far more permissions (Listing 1.5) than it really needs, which is a trend often followed by Android Apps, in general [28]. We notice that the App asks for permissions to access the user's exact position and the user has no other choice than agreeing in order to install and use the App. This is acceptable for an application meant to facilitate the use of public transportation. However the user grants this permission to the App which does not imply that the user also accepts this information to be sent to a unknown third party server, with no information on who will store and use this data, and for what purposes. **The Android permission system cannot be interpreted as an informed end-user agreement for the collection and use of personal data by third-parties**.

4 Related Works

In this domain of smartphones and privacy, PiOS [26] (in case of iOS) and TaintDroid [27] (in case of Android) are two major contributions. They rely on totally different approaches, since PiOS employs static analysis of the App binaries, whereas TaintDroid uses a dynamic taint analysis that requires modifying the Dalvik Virtual Machine. Recent work by Han et al. [30] compares and examines the difference in the usage of security/privacy sensitive APIs for Android and iOS. Their analysis revealed that iOS Apps access more privacy-sensitive APIs than Android Apps: as mentioned in the paper, this is probably due to the

absence of end user notifications with the iOS version the authors used. However, since the introduction of iOS6, a user permission is solicited the first time an App tries to access private data (Contacts, Location, Reminders, Photos, Calendar and Social Networking accounts). Later, iOS remembers and follows the user preferences, whereas also allowing the user to change his preferences at any time. [3] discusses Androguard, a tool that can be used for reverse engineering and malware analysis of Android Apps. In addition, mobilescope was a tool that has recently been acquired by Evidon and included in their product Evidon Encompass [16]. This tool analyzes the network traffic using a man-in-the-middle (MITM) proxy to detect privacy leaks. From this point of view, our approach is better as more and more Apps can detect MITM proxies and stop working if one is found (e.g. Facebook). Also, the user needs to know in advance what data to look for in the network. Furthermore, our instrumented Android system not only uses TaintDroid [27] but also looks for the private data in the network traffic leaving the device. This enables us to detect the private data leakage even if TaintDroid is not able to detect it, which is often the case [29]. In the iOS world, there has been two other works, namely PMP [25] and PSiOS [31], but yet again, they don't provide any insight about the potential private data leakage over the network: they just deal with mere access to private data. PSiOS is a system designed for iOS that enables fine-grained policy enforcement but it is also limited when it comes to the real detection of private data leakage. The simple access to private data and its transmission over the network are two different things. Our system is even able to capture to which server the data is actually sent and thereby, eventually be able to distinguish between first and third-party. Also, as our analysis is essentially at App run-time, obfuscation techniques can no longer be used to bypass the detection of private data leakage.

5 Conclusions

This article discusses bad practices employed in the world of smartphones and the limitations of the privacy control features proposed by Android/iOS Mobile OSs. The RATP App (iOS 5.4.1 and Android 2.8 versions) provides a good illustration of bad practices by some companies as many kinds of private data are collected and sent, and in this example, even if the "legal terms" of the RATP App claims the contrary.

Android decided to use a user-centric permission system, for the moment only at installation time, to let the end-user decide whether or not he/she grants specific permissions to an application (this may change very soon). Obviously this system does not help so much in controlling what information is captured and sent: it is a coarse-grained system that works in binary mode, without any behavioral analysis of the App and without making any difference between communications to first or third-party servers. In other words, the system has limited benefits from the end-user point of view.

On the opposite Apple chose to follow market-level checks, plus user-centric control through a dedicated privacy dashboard, as well as restrictions (UDID

ban in iOS6, MAC address access in iOS7) associated to incentives to follow good practices (Advertising ID). In this work we show how A&A companies have found ways, and even specifically designed techniques to bypass some of these restrictions. For instance, when the UDID was deprecated and replaced by an Advertising ID, an OpenUDID service appeared to provide a similar feature (and this OpenUDID service is used by the RATP App). We also show that other types of permanent (or at least long term) identifiers are accessed and transmitted to remote A&A servers (WiFi MAC address, device name, UDID) in case of the RATP App. The collection of such stable identifiers is highly useful to A&A companies: a user may reset its Advertising ID as often as he/she wants, this has no impact on the ability of A&A companies to continue tracking this device. We cannot imagine that Apple is not aware of the situation.

All of this is happening without the user knowledge and perhaps without the App developer's knowledge (were not considering the particular case of the RATP App here). An App developer often includes an advertising library without knowing its behavior, and if there is no legal risk in case important privacy leaks are discovered in his App, this developer will probably not care too much.

NB: the RATP has published an answer to our findings. This answer and our initial blog can be found at [19]:

References

1. A guide on how to decrypt iOS App binaries. http://rce64.wordpress.com/2013/01/27/private-decrypting-apps\-on-ios-6-multiple-architectures-and-pie/.
2. Adgoji: A mobile analytics company. http://www.adgoji.com/.
3. Androguard. http://code.google.com/p/androguard/.
4. Apple Documentation on sysctl function. http://developer.apple.com/library/ios/#documentation/system/conceptual/manpages_iphoneos/man3/sysctl.3.html.
5. Apple Documentation on UIApplication Class. http://developer.apple.com/library/ios/#DOCUMENTATION/UIKit/Reference/UIApplication_Class/Reference/Reference.html.
6. Apple Documentation on URLScheme. http://developer.apple.com/library/ios/#featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007891-SW1.
7. Apple: iOS 7 returns the same value of MAC address. https://developer.apple.com/news/?id=8222013a.
8. Blog: RATP Android App leaking private data in clear. http://www.rfc1149.net/blog/2012/03/20/a-qui-la-ratp-\\vend-elle-nos-informations-personnelles/.
9. Classdumpz. http://code.google.com/p/networkpx/wiki/class_dump_z.

10. Dshield tools. https://secure.dshield.org/tools/.
11. Flurry: An analytics company. http://www.flurry.com/flurry-analytics.html.
12. IDA Pro. https://www.hex-rays.com/products/ida/index.shtml.
13. Infosniper IP Geolocalisation service. http://www.infosniper.net/.
14. iOS 6 Privacy Extension package. http://planete.inrialpes.fr/~achara/mobilitics/iOS6PrivacyExtension.html.
15. iOS Sandboxing: Apple Documentation. http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/TheiOSEnvironment/TheiOSEnvironment.html.
16. MobileScope. http://www.evidon.com/mobilescope.
17. MobileSubstrate. http://iphonedevwiki.net/index.php/MobileSubstrate.
18. OpenUDID: Alternative to UDID on iPhone. http://www.flurry.com/flurry-analytics.html.
19. RATP blog, Privatics team, Inria. http://goo.gl/dyurp.
20. Smartphone OS Market Share. http://www.idc.com/getdoc.jsp?containerId=prUS24108913.
21. Sofialys. http://www.sofialys.com/en/.
22. Sofialys icon. http://88.190.216.131/favicon.ico.
23. UIPasteboard Apple Documentation. http://developer.apple.com/library/ios/#documentation/uikit/reference/UIPasteboard_Class/Reference.html.
24. Whois service. http://en.wikipedia.org/wiki/Whois.
25. Y. Agarwal and M. Hall. ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing. In *Proceedings of the ACM International Conference on Mobile Systems, Applications and Services (MobiSys), Taipei, June 2013.*
26. M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS : Detecting privacy leaks in iOS applications. In *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, San Diego, CA, USA*, February 2011.
27. W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, October 2010.
28. A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security (CCS)*, 2011.
29. Golam Sarwar and Olivier Mehani and Roksana Boreli and Dali Kaafar. On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices. In *SECRYPT 2013, 10th International Conference on Security and Cryptography*, July 2013.
30. J. Han, Q. Yan, D. Gao, J. Zhou, and R. H. Deng. Comparing Mobile Privacy Protection through Cross-Platform Applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2013.
31. Tim Werthmann and Ralf Hund and Lucas Davi and Ahmad-Reza Sadeghi and Thorsten Holz. PSiOS: Bring Your Own Privacy & Security to iOS Devices. In *8th ACM Symposium on Information, Computer and Communications Security (ASIACCS), May, 2013.*

6 Some screenshots

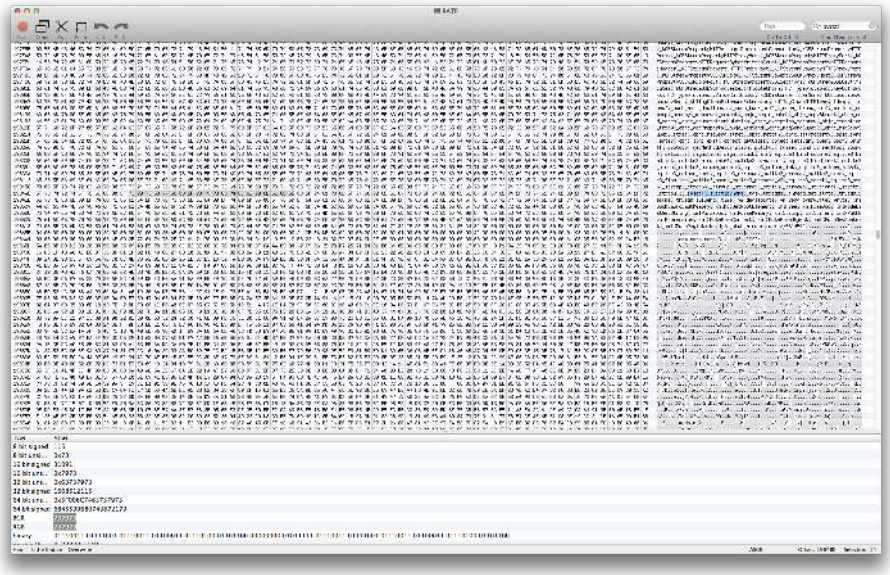


Fig. 2. RATP Binary opened in a hexeditor

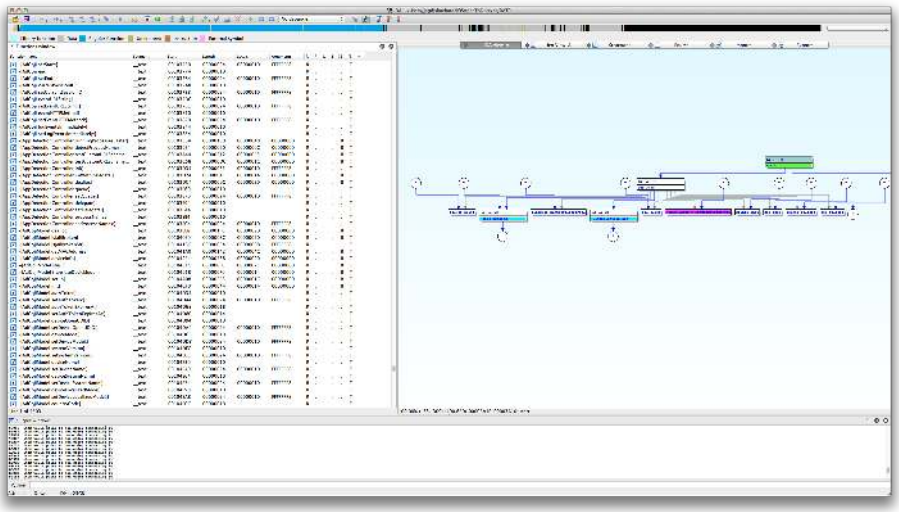


Fig. 3. RATP Binary opened in IDA

Fig. 4. Headers generated by class-dump-z

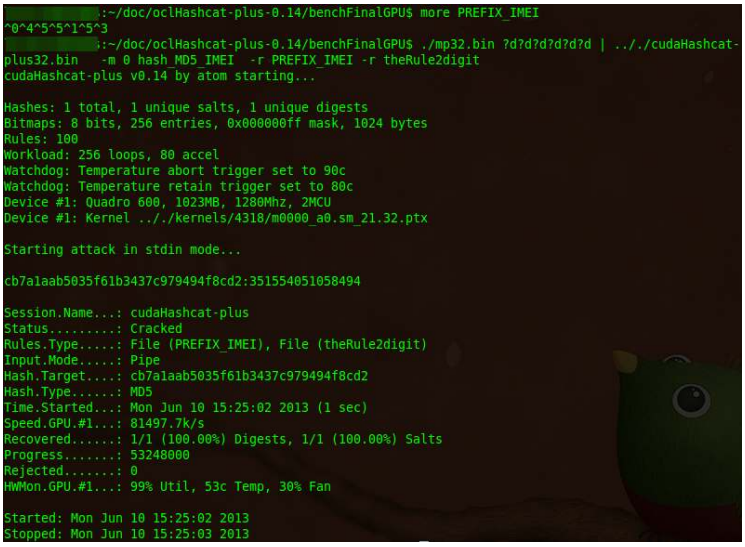


Fig. 5. IMEI Deanonimyzation given information about device

```
Class descriptor  : 'Lnet/hockeyapp/android/UpdateActivityInterface;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateInfoAdapter$1;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateInfoAdapter;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateInfoListener;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateManager;'
Class descriptor  : 'Lnet/hockeyapp/android/UpdateManagerListener;'
Class descriptor  : 'Lnet/hockeyapp/android/VersionCache;'
Class descriptor  : 'Lcom/adbox/AdBoxLibrary$6;'
Class descriptor  : 'Lcom/adbox/AdBoxLibrary;'
Class descriptor  : 'Lcom/adbox/beans/BanniereDynamique;'
Class descriptor  : 'Lcom/adbox/beans/BanniereExtensible;'
Class descriptor  : 'Lcom/adbox/beans/BanniereRetractable;'
Class descriptor  : 'Lcom/adbox/behavior/DynamicAdBehavior;'
Class descriptor  : 'Lcom/adbox/display/DisplayRetractableBanner;'
Class descriptor  : 'Lcom/adbox/imgthread/ImgException;'
Class descriptor  : 'Lcom/adbox/parsethread/ParseException;'
Class descriptor  : 'Lcom/fabernovel/ratp/AbstractWebMapActivity;'
Class descriptor  : 'Lcom/fabernovel/ratp/AlertingActivity;'
Class descriptor  : 'Lcom/fabernovel/ratp/DetailsTrafic;'
Class descriptor  : 'Lcom/fabernovel/ratp/DetailsTravaux;'
Class descriptor  : 'Lcom/fabernovel/ratp/FDRoute;'
Class descriptor  : 'Lcom/fabernovel/ratp/HorairesResultats;'
Class descriptor  : 'Lcom/fabernovel/ratp/PlansAffichage$StationsOverlay;'
Class descriptor  : 'Lcom/fabernovel/ratp/ProximitePlan$StationsOverlay;'
Class descriptor  : 'Lcom/fabernovel/ratp/Trafic;'
Class descriptor  : 'Lcom/fabernovel/ratp/entity/BusStop;'
Class descriptor  : 'Lcom/fabernovel/ratp/entity/Station;'
```

Fig. 6. Listing class descriptors of Android App

Fig. 7. Listing AdBox Headers in iOS binary of RATP App