WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission

Jagdish Prasad Achara, Mathieu Cunche, Vincent Roca, Aurélien Francillon

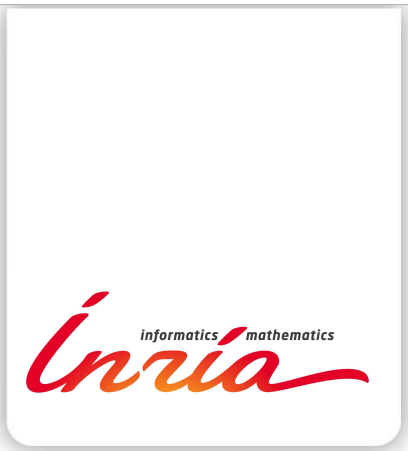
WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission

Jagdish Prasad Achara[†], **Mathieu Cunche**[†*], **Vincent Roca**[†], **Aurélien Francillon**[‡]

[†] **Inria, PRIVATICS team, France** *firstname.name@inria.fr*
[*] **INSA-Lyon, CITI lab., France**
[‡] **EURECOM, France** *firstname.name@eurecom.fr*

RESEARCH REPORT
N° 8539
May 2014
Project-Team PRIVATICS

ISSN 0249-6399 ISRN INRIA/RR--8539--FR+ENG



WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission

Jagdish Prasad Achara[†], Mathieu Cunche[†*], Vincent Roca[†], Aurélien Francillon[‡]

[†] Inria, PRIVATICS team, France *firstname.name@inria.fr*
[*] INSA-Lyon, CITI lab., France
[‡] EURECOM, France *firstname.name@eurecom.fr*

Project-Team PRIVATICS

Research Report n° 8539 — May 2014 — 21 pages

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Abstract: On Android, users can choose to install an application, or not, based on the permissions it requests. These permissions are later enforced on the application by the system, e.g., when accessing sensitive user data. In this work, we focus on the access to Wi-Fi related information, which is protected by the `ACCESS_WIFI_STATE` permission. We show that this apparently innocuous network related permission can leak Personally Identifiable Information (PII). Such information is otherwise only accessible by clearly identifiable permissions (such as `READ_PHONE_STATE` or `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION`). We analyzed permissions of 2700 applications from Google Play, and found that 41% of them use the `ACCESS_WIFI_STATE` permission. We then statically analyzed 998 such applications and, based on the results, selected 88 for dynamic analysis. Finally, we conducted an online survey to study the user perception of the privacy risks associated with this permission. Our results demonstrate that users largely underestimate the privacy implications of this permission, in particular because they often cannot realize what private information can be inferred from it. Our analysis further reveals that some companies have already started to abuse this permission to collect personal user information, for example, to get a unique device identifier for tracking across applications or to geolocalize the user without explicitly asking for the dedicated permissions. Because this permission is very common, most users are potentially at risk. There is therefore an urgent need for modification of the privileges granted by this permission as well as a more accurate description of the implications of accepting a permission.

Key-words: Android Permissions, Wi-Fi related data, Privacy, Personally Identifiable Information (PII) leakage, Static and Dynamic Analysis of Applications, User survey

WifiLeaks: Sous-estimation des implications en termes de vie privée de l'autorisation ACCESS_WIFI_STATE d'Android

Résumé : Avec Android, les utilisateurs peuvent choisir d'installer ou non une application en fonction des permissions demandées par cette dernière. Ces permissions sont ensuite imposées à l'application par le système d'exploitation, par exemple lors de l'accès à des données sensibles de l'utilisateur. Dans ce travail nous nous intéressons à l'accès aux informations relatives au Wi-Fi, accès protégé par la permission `ACCESS_WIFI_STATE`. Nous montrons que cette permission de type réseau et d'apparence très anodine, peut être la cause de fuites d'informations personnelles (PII), qui ne seraient sinon accessibles que par des permissions clairement identifiables (telles que `READ_PHONE_STATE` ou `ACCESS_FINE_LOCATION` ou `ACCESS_COARSE_LOCATION`). Nous avons analysé les permissions de 2700 applications du marché Google Play, et nous avons trouvé que 41% d'entres elles demandent la permission `ACCESS_WIFI_STATE`. Nous avons ensuite analysé de façon statique 998 applications de cet ensemble, et en fonction des résultats, nous en avons sélectionné 88 pour une analyse dynamique plus poussée. Finallement nous avons conduit une enquête en ligne pour étudier la perception qu'ont les utilisateurs des risques associés à cette permission. Nos résultats démontrent que les utilisateurs sous estiment largement les implications en termes de vie privée de cette permission, en particulier parce qu'ils ne peuvent pas réaliser quelles informations privées peuvent en être tirées. Nos analyses montrent par ailleurs que certaines sociétés ont commencé à abuser de cette permission pour collecter des informations personnelles, par exemple pour obtenir un identifiant unique et stable du terminal à des fins de traçage, ou pour géolocaliser l'utilisateur sans avoir à lui demander explicitement l'autorisation. Parce que cette permission est très répendue, la plupart des utilisateurs courrent potentiellement un risque. Il y a donc un besoin urgent de modifier les privilèges associés à cette permission ainsi que de décrire plus précisement les implications que son acceptation peut avoir.

Mots-clés : Permissions Android, données relatives au Wi-Fi, Vie privée, fuites d'Information Personnellement Identifiables (PII), Analyse statique et dynamique d'applications, Enquête utilisateurs

1 Introduction

Mobile devices have become ubiquitous and are crucial part of our lives today. This is mainly due to the wide variety of functionalities they provide beyond telephony and Internet services, being equipped with a lot of different kinds of sensors: e.g., GPS navigation unit, Camera, Accelerometer. As these devices know the most about their users, they've invariably become the most serious threat to user privacy invasion. Above all, since a better user profile can be created with the help of vast amount of data available and accessible, Advertising and Analytics (A&A) companies have shifted their focus from traditional desktop computers/browsers to applications running on these devices.

Today a large fraction of mobile devices are running the Android OS. To control the access to user data, it implements a permission system where a user needs to grant permissions required by an application at installation time. The goal of this permission system is to let a user know in advance the information an application would be able to access if installed. It is worth mentioning that an application can only be installed if the user agrees with the required list of permissions.

However, this permission system has shown limitations. Due to a poor understanding of the permission descriptions and a lack of attention given to these messages, many applications are installed without the user really knowing about the kind of information applications would be able to access [16]. In addition, the permission system sometimes fails to prevent the access to sensitive data [23] allowing applications to access information without the need of corresponding permission.

In this paper we focus on a peculiar permission, `ACCESS_WIFI_STATE`, that allows an application to access various information related to the the Wi-Fi interface. This permission falls in the 'network' group of permissions [3] and is categorized as 'normal' (as compared to 'dangerous', for example) based on the potential risk implied in the permission [2] and thereby most users probably fail to see associated privacy risks.

In this paper we show that the reality is just opposite and that a plenty of user PII can be (and are already) derived from the data accessible thanks to this permission. This is a severe problem as the use of `ACCESS_WIFI_STATE` permission by advertising libraries has increased over the time [10].

More specifically the contributions of this work are threefold:

1. First, we consolidate what PII can be derived from the data related to Wi-Fi interface (Section 3), namely unique and stable identifiers (useful for tracking purposes), device geolocation, travel history and social links between users. In addition, we also provide details about the uniqueness of SSIDs by analyzing a worldwide database of free and paid public Wi-Fi APs maintained by `jiwire.com`.

2. Then we analyze the current situation on the Google Play employing both a static and dynamic analysis of Android applications (Section 4). We first show that a large proportion of applications (41% of the 2700 Apps we analyzed) ask for this permission. While asking for this permission can be justified in some categories of applications, it is not at all the case in others. Going further into the details, we also reveal that a large number of both first and third-parties have actually started to exploit this permission by directly accessing or deriving various user PII from the Wi-Fi related data accessible to them thanks to this permission.

3. Finally, we analyze the user perception of this permission using an online survey to which 156 Android users answered (Section 5). The results clearly demonstrate that users don't really understand the privacy implications of this permission. Changes in Android permission system are solicited to tackle this problem, for example, either by modifying the list

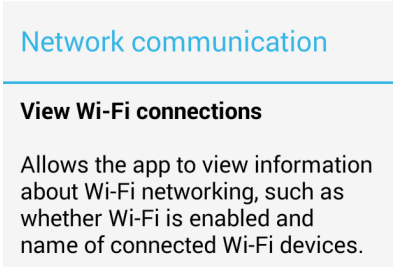


Figure 1: Description of the `ACCESS_WIFI_STATE` permission as presented to a user.

of methods restricted by the permission or by displaying the appropriate message to the user.

2 Background

2.1 Android permission system

The Android security architecture has two levels of protection: first, all the applications and some parts of the system run as different Unix users, and secondly, applications must explicitly ask the user for permissions.

The first level of protection prevents applications to sneak into each other or into the system data/activity, since each application runs in its own process sandbox. The second level of protection comes into play as, by default, Android gives no privilege to applications. As Android gives no privilege to applications by default, the second level of protection comes into play in the form of permissions. Each application must ask the user for privileges by statically declaring the list of permissions it requires. At installation time, the Android system prompts the user for consent, and the installation completes only if the user agrees with it.

There are a total of 145 different permissions available (as of Android version 4.4) for an application to ask for. Many of these permissions are required by applications to access user sensitive information (e.g. `ACCESS_FINE_LOCATION` and `READ_CONTACTS` are the permissions required to respectively geolocalize the device and read user's contact data). The permissions are also categorized, based on the associated potential risks, as either 'normal', 'dangerous', 'signature', or 'signatureOrSystem'.

Since the kind of information exposed by a particular permission is not necessarily trivial for the end user, each permission is accompanied by a short description. Figure 1 presents the description of the `ACCESS_WIFI_STATE` permission as displayed to the user. This message contains a title, a subtitle and a short description. The blue title indicates a broad categorization of the permission, the subtitle is the text corresponding to the name of the permission, and the description gives a short explanation of the privileges granted by the permission. Based on this information, users are expected to be able to be able to decide whether a permission is really required by an application.

2.2 The `ACCESS_WIFI_STATE` permission

The `ACCESS_WIFI_STATE` permission, falls in the group of 'Network communications'. It grants access to the data associated with the Wi-Fi interface, making some of the methods of the `WifiManager` class [7] available to the application. These methods and their capabilities are

Table 1: `WifiManager`'s method restricted by the `ACCESS_WIFI_STATE` permission.

Method name	Description	Retrievable Information
isWifiEnabled()	Returns whether Wi-Fi is enabled or disabled.	Returns true if Wi-Fi is enabled.
getWifiState()	Gets the Wi-Fi enabled state.	Enabled, disabled, currently being enabled or disabled, unknown
getConfiguredNetworks()	Returns a list of all configured networks.	**For each configured network/AP**: SSID, allowed protocols and security schemes
getConnectionInfo()	Returns dynamic information about the current Wi-Fi connection, if any is active.	**About AP:** BSSID, SSID, RSSI **About Device:** Wi-Fi MAC address, IP address
getScanResults()	Returns the results of the last AP scan.	**For each AP:** BSSID, SSID, signal strength, channel, capabilities
getDhcpInfo()	Returns the DHCP-assigned addresses from the last successful DHCP request, if any.	IP address, DNS server address, gateway and netmask

presented in table 1. It is worth mentioning that these methods only allow to read the data associated with the Wi-Fi interface, but not to modify it: a different permission, `CHANGE_WIFI_STATE`, is required to change the state of the Wi-Fi interface.

After analyzing the potential risks implied by this permission, Google decided to characterize it as 'normal' (we will see that this decision is highly questionable). Also, it is important to realize that this is not the permission required for Internet access. `INTERNET` is the only permission required to open a network socket to send or receive data over the network.

3 Inference of User PII from Wi-Fi related data

As we have seen, the `ACCESS_WIFI_STATE` permission enables an application to read data related to the Wi-Fi configuration of the device. This raw data may look inoccuous, but it is actually possible to infer several user PII. In this section we describe such user PII that can be either directly accessed or derived from raw data related to Wi-Fi.

3.1 A unique device identifier

Using the `getConnectionInfo()` method, an application can obtain the MAC address of the Wi-Fi interface, a 48-bit identifier that is unique to a device. A unique identifier permanently attached to a device allows external entities to track user activities across all applications. And since a single user is usually associated with a given device, it also enables to identify the user, which explains why the Wi-Fi MAC address is considered as a user PII.

In fact, the Wi-Fi MAC address is not the only hardware-tied identifier available to be accessed by applications on Android: for example applications having the `READ_PHONE_STATE` permission can access the `IMEI` and `MEID`. But the Wi-Fi MAC address is somehow unique among all hardware-tied identifiers as it can also be used by trackers to link both the online and physical profiles of the user. The Wi-Fi MAC address is used by tracking systems that monitor each wireless channel to track individuals as they move in the physical world (e.g., in a retail store of a shopping mall) [21, 14]. If in parallel it is also collected by applications, then it paves the way for trackers to create a bigger profile based on both the online and physical activities of the

user. Indeed, this technique can be used to detect the user and to serve targeted advertisement in the physical world based on the online profile of the user [6]. It is quite evident from the above discussion that the availability of Wi-Fi MAC address to advertisers is a serious threat to the user privacy.

3.2 Geolocation

Geolocation is traditionally obtained using the GPS navigation unit consisting of a dedicated GPS chip embedded in the device. However, Wi-Fi and GSM based geolocation systems have seen wide deployment during last few years either to improve accuracy and decrease the first fix time of the geolocation or as an alternative to the GPS. These systems use the information about surrounding Wi-Fi APs (resp. GSM cell towers) to derive device geolocation.

The list of surrounding Wi-Fi APs can be obtained thanks to the `ACCESS_WIFI_STATE` permission through the `getScanResults()` method of `WifiManager` class. For each access point in range, the retrieved information includes: the BSSID (MAC address of the AP), the SSID (human readable name), received signal strength, supported encryption schemes, operating channel, etc. However, it is very important to mention here that a calling the getScanResults() method does not launch the scanning of surrounding Wi-Fi APs, instead it returns the list of last scanned Wi-Fi APs. As the list retrieved by the application might not be up-to-date, an application might not always be able to derive the current device location. However, in practice, we notice that Android system is doing the scan of surrounding Wi-Fi APs every 15 seconds[1]. Moreover, there might be other applications or system components that trigger the scan in the meantime. Therefore in practice the list of surrounding Wi-Fi APs is often up-to-date and an application does not really need to start the scanning by itself.

By submitting the raw result of a Wi-Fi scan to a remote geolocation service (companies like Google [5], Microsoft, Skyhookwireless and Apple, to name a few, provide such a service), the device gets in return geolocation information (coordinates and accuracy metric). And the precision in case of Wi-Fi based geolocation is quite accurate in cities: around 20 meters in urban areas [18]. This is due to both the high density of Wi-Fi APs in urban areas and a coverage area comparatively smaller than GSM Cell towers.

The Wi-Fi, GSM and GPS-based geolocation systems are actually employed by the Android system. However applications can only access the geolocation provided by Android system if they have `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` permissions that a user easily understands.

On the opposite, the raw Wi-Fi scan information is a source of geolocation information that is not protected by any of these two geolocation permissions. It is therefore possible for an application to obtain geolocation information without having to explicitly ask for it through the official `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` permissions. An application can retrieve the raw Wi-Fi APs scan and query the remote geolocation service provided that it is granted both the `ACCESS_WIFI_STATE` permission and the `INTERNET` permission. And we will see in Section 4 that applications are often granted both the `ACCESS_WIFI_STATE` and `INTERNET` permissions.

[1]We checked Android 4.1.1 and 4.4 source. The value of 'config_wifi_supplicant_scan_interval' present in './frameworks/base/core/res/res/values/config.xml' file is 15000 milliseconds. Of course, this value is subject to be changed later by the manufacturers or carriers while distributing a modified copy of Android OS along with the device. It can be either changed directly in the Android source or it can be set in device 'build.prop' file.

3.3 Travel history

The list of Wi-Fi networks to which the device has been connected is stored in the *Configured Networks List* that can be accessed through the `getConfiguredNetworks()` method of `WifiManager` class. For each of these configured networks, the SSID is available along with other information such as the supported security protocols and authentication algorithms. And data stored in the *Configured Networks List* can be combined with external ressources to obtain information such as the previously visited locations.

Indeed Wi-Fi AP mapping databases exist that store the geolocation coordinates of Wi-Fi APs, along with other information [8]. The identifiers stored in the *Configured Networks List* can be used to retrieve geolocation information from those databases [17]. In fact, only the SSID of the configured networks seems to be available through the `getConfiguredNetworks()` method[2], and contrary to the BSSID it is not a unique identifier of Wi-Fi AP. Therefore, multiple Wi-Fi APs can share the same SSID across the world potentially reducing the accuracy of the geolocation information obtained from those databases.

To evaluate the accuracy of the SSID-based geolocation, we have analyzed a database of free and paid public Wi-Fi hotspots spread over 123 countries. This database, obtained from the Wi-Fi Finder application (from `jiwire.com`) has 5,28,994 entries of 3,16,792 distinct Wi-Fi hotspots. Among these, 2,92,296 (92%) Wi-Fi hotspots have a unique SSID whereas 24,496 have not. This clearly shows that most public Wi-Fi hotspots (both free and paid) have unique SSIDs, ensuring that SSIDs of Wi-Fi hotspots found in the *Configured Networks List* can be linked, most of the time, to a unique location.

3.4 Social links

Identifiers stored in the *Configured Network List* can also be used to infer ties between individuals. Indeed, it has been shown that by comparing the list of SSIDs of *Configured Network List* on two devices, it is possible to predict the existence of a social or professional link between the owners of those devices [13]. By collecting this data on a large population, an application could gather information that would make it possible for them to build a social network between their users.

The work presented in [13] relies only on the SSID, which is not a unique identifier and can therefore lead to some limitations. However, the configured network list returned by the Android system also contains some other information about the configured Wi-Fi AP such as allowed protocols, authentication algorithms, key management, etc. This information could be leveraged to improve the quality of the social link predictor.

3.5 Other PII derived from SSIDs

Untill this point, we have considered the SSID as a genuine identifier. We now consider the possibility of extracting additional information from the names of these SSIDs themselves. Indeed, SSIDs are strings of characters designed to be meaningful for users and potentially contain information about the network owner or its users. More specifically, SSIDs often designate entities such as institutions, individuals or locations [19]. Extraction of this information can be done manually or could be automated.

Automatically extracting information from those SSIDs is close to the problem of the Named Entity Recognition [12]. It is a subtask of information extraction to classify atomic elements in text into predefined categories, such as, the names of persons, organizations and locations.

[2]We wrote a test application on Android 4.4 and find that BSSID field is returned as null by the system. However, as per Android Doc, actual BSSID of Wi-Fi AP should be returned if it is set.

SSIDs could be analyzed by such tools developed in this field to extract information as it is done in [20] for short messages found on Twitter. This information can then be used, for example, by advertisers to enrich the profile of the user to serve targeted ads.

4 Android Applications Analysis

We now analyze 100 most popular free applications in all 27 categories present on Google Play, i.e., 2700 applications in total. We start by crawling Google Play using an unofficial API [4] and look at the list of permissions required by these applications. For further analysis, we only select applications that asks for both `ACCESS_WIFI_STATE` and `INTERNET` permissions and find that a significant proportion of applications (1101 applications or 41% of 2700 applications) require both of them. In practice, we find that almost all applications requiring `ACCESS_WIFI_STATE` permission also ask for `INTERNET` permission[3].

We first statically analyze 998 APK files that we were able to download among 1101 applications requiring both permissions[4] and then, only a subset of those applications are chosen for dynamic analysis.

4.1 Static analysis

We use Androguard [1] for static analysis and our own Python scripts. Androguard is an open source reverse engineering tool that is capable of analyzing Android APK files.

Table 2: Most commonly accessed methods of `WifiManager` class, in 998 applications.

Method call	**# of Apps**
getConnectionInfo()	753 (75.45%)
isWiFiEnabled()	344 (33.47%)
getScanResults()	156 (15.63%)
getConfiguredNetworks()	59 (5.91%)
getWifiState()	76 (7.62%)
getDhcpInfo()	63 (6.31%)

As described in Section 2.2, `ACCESS_WIFI_STATE` permission is required to access various methods related to Wi-Fi state and configuration. Table 2 presents the number of applications accessing these methods. Indeed we find that 165 (∼17%) applications have just put this permission in their list of required permissions without really accessing any of the methods protected by this permission. Actually these overprivileged applications also present a privacy-risk to the users as their future revisions could easily exploit the resources protected by this permission.

Among these 6 methods protected by this permission, we chose to focus on the ones who pose serious privacy risks to the user, namely getScanResults(), getConfiguredNetworks() and getConnectionInfo(). The other three remaining methods are not privacy sensitive as they give either information on the status of the Wi-Fi interface (getWifiState() and isWiFiEnabled()) or information about the local IP configuration of the Wi-Fi interface (getDhcpInfo()). The three chosen methods represents a privacy threat because, as seen in Section 3, they give direct or indirect access to a number of user PII. With the help of getConnectionInfo() method call, an application can access one of the device unique identifier, i.e., MAC Address of Wi-Fi chip). In

[3]There are only 5 applications who require `ACCESS_WIFI_STATE` permission but not `INTERNET` permission.
[4]103 APKs could not be downloaded due to some technical hurdle.

Table 3: Correlation between privacy-sensitive methods of `WifiManager` class (out of 762)

Method calls	# **Apps**
(getConnectionInfo, getConfiguredNetworks)	33
(getConfiguredNetworks, getScanResults)	19
(getScanResults, getConnectionInfo)	62

the same way, getScanResults() returns the list of last scanned Wi-Fi APs that can be turned into geolocation information with the aid of geolocation service. The last method, i.e., getConfiguredNetworks() returns the SSIDs among other information of all the APs to whom the user has already connected or configured at some point of time in the past. As described in Section 3, even if the returned list of last scanned surrounding Wi-Fi APs is almost up-to-date, we find that 196 applications (out of 1101 applications with `ACCESS_WIFI_STATE` and `INTERNET` permission) also require `CHANGE_WIFI_STATE` permission and 136 applications explicitly start the scanning before accessing this list of surrounding Wi-Fi APs.

From now on, our analysis focuses only on these three methods. Table 3 presents the correlation between these methods based on the number of applications accessing different combinations of these privacy sensitive methods. We can see that the most correlation exists between getScanResults() and getConnectionInfo() methods. Otherwise, there are 762 applications (∼76%, out of 998 applications tested) accessing, at least, one of these three privacy sensitive methods whereas 11 applications are accessing all 3 privacy sensitive methods.

In fact, we might expect applications in some categories to access these methods; for example, it is expected that a Wi-Fi manager application (in Tools or App Widget categories) access these methods, but not a cooking or wallpaper application. Figure 2 presents a category-wise distribution of number of applications accessing these privacy-sensitive APIs out of a total of 100 applications analyzed in each category. This category-wise distribution is important to realize why applications in certain categories would need access to `ACCESS_WIFI_STATE` permission.

It is the applications in 'Game' category that access the most information about currently connected Wi-Fi APs through *getConnectionInfo()* method. It is justifiable in some cases, for example, the networked game applications. Otherwise, for applications of all other categories, there doesn't seem to be any obvious reason to do so. Once again, it's the applications in 'Game' category that access the most the list of surrounding Wi-Fi APs through *getScanResults()* method. This time also, it could be justifiable in case of networked games. But the second category in this list is 'Social' applications, which seems to be somewhat abnormal at first glance. But as this information could be used to geolocalize the user, we speculate it as the potential reason behind its access. Lastly, information about configured networks is most accessed by applications in 'Tools' category which is, in fact, expected but is not justifiable at all, for example, by applications in 'Shopping' category. Only one other category from which we can imagine applications accessing this information is 'App Widgets'. Overall (combining access of all three methods), applications in 'Game' category are accessing the most these 3 privacy-sensitive methods which is suspicious. Specifically, we see applications in some categories like lifestyle, comics, app wallpaper, medical, finance, games, etc. accessing all three methods but, from whom one might not expect at all to access these methods.

4.1.1 Access of privacy-sensitive methods by third-party code present inside applications

During the analysis of the applications, we have identified that the method calls were either made by code written by the application developer (considered as first-party) or by the code

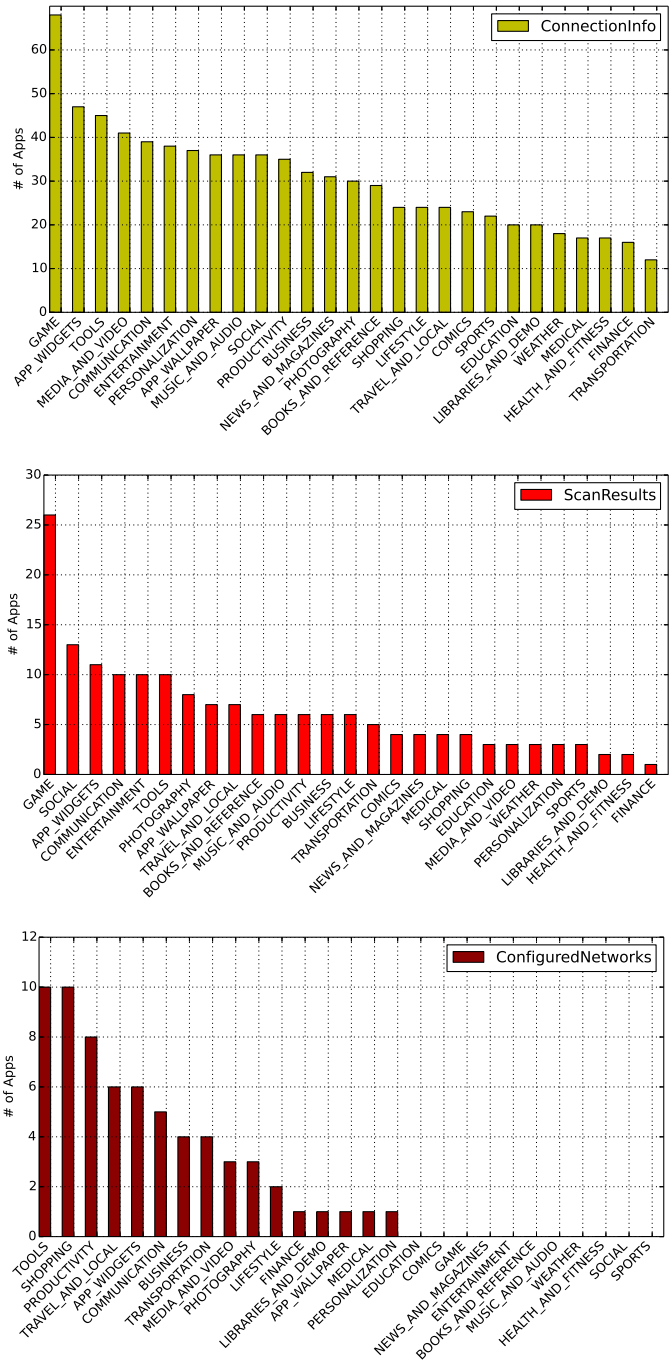


Figure 2: Google Play category-wise distribution of applications based on 3 privacy-sensitive method calls out of a total of 100 applications in each category.

from third-party libraries (for example, advertisement, analytics, performance monitors, crash reporters) included by the application developer. We therefore study the origin of the calls made to these methods: if it made by code from first party or third party. If the class accessing these various methods belong to a package whose name is the same as the application package name, we mark it as the code written by the application developer; otherwise, it is considered as third-party code.

Among these 762 applications who access at least one of these methods, 136 (i.e., 18%) applications are such that it is only due to the third-party code present inside them. It highlights the fact that this permission is being used by third-parties than the application developer in a significant number of applications. We note that the access to an API call by third-party code might be completely justified in some situations. However this is not always the case and for instance `inmobi.com` and `skyhookwireless.com` call the Wi-Fi related methods in many applications and we do not believe that it is justified.

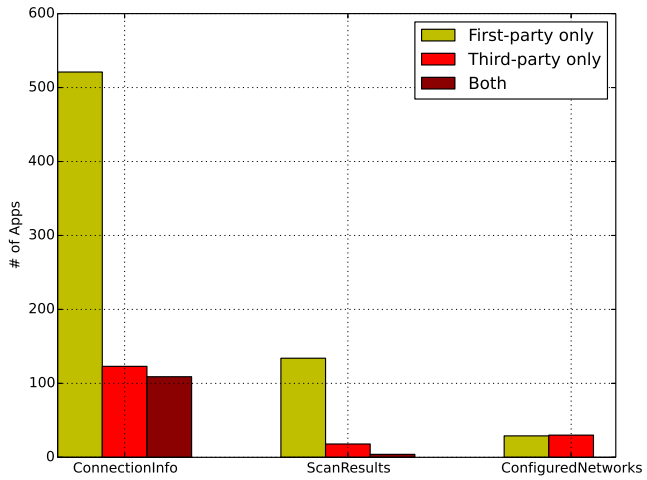


Figure 3: Distribution of applications based on the party accessing privacy-sensitive methods out of a total of 762 applications.

Figure 3 presents, for each of the three privacy-sensitive methods, the distribution whether a first, third or both parties access these methods, and reveals that there are some applications in which only the third-party code accesses these privacy-sensitive methods. This means that the code written by the application developper in itself doesn't require `ACCESS_WIFI_STATE` permission but it is, in fact, due to the third-party library code present inside the application. When those method calls are made by a third party library, this can either be to secretly collect information on the users or to provide a functionality to the application; for example, an application developer or a third-party can use the code from `skyhookwireless.com` to retrieve device geolocation without needing explicit dedicated geolocation permissions. It is worth mentioning that `skyhookwireless.com` retrieves the list of surrounding Wi-Fi APs in 5 applications (Table 4). In all cases, no matter first or third-party, deriving device geolocation without explicit user permission is not legitimate and should be protected by the Android system. This also raises the question of whether these third-parties ask application developers to put this permission in their required list of permissions or they just make use of it when third-party code is included inside an application requesting this permission. We cannot definitively know the answer of this question but it's possible that a geolocation service provider can just ask the developer to request `ACCESS-`

`_WIFI_STATE` and `INTERNET` permissions, and assure the application developer to provide the geolocation when requested.

Table 4: Top 10 third-parties in each category and their corresponding number of applications.

ConnectionInfo		**ScanResults**		**ConfiguredNetworks**	
Third-party Third-party	**# Apps**	**Third-party Third-party**	**# Apps**	**Third-party Third-party**	**# Apps**
inmobi.com	74	inmobi.com	99	google.com	10
chartboost.com	55	domob.cn	9	mobiletag.com	4
tapjoy.com	49	mologiq.com	6	lechucksoftware.com	2
vungle.com	47	tencent.com	5	android.com	2
jirbo.com	43	skyhookwireless.com	4	Unibail.com	1
mobileapptracker.com	42	mobiletag.com	4	gpit.com	1
startapp.com	33	life360.com	2	citrix.	1
searchboxsdk.com	32	baidu.com	2	neom.de	1
crashlytics.com	30	mapbar.com	2	mcafee.com	1
amazon.com	29	Unibail.com	1	ookla.com	1

Table 4 presents top 10 third-parties in each category and their corresponding number of applications in which they are present. Looking at the webpages of these third-parties, one may understand the purpose of these third-parties in various applications. It seems like most of them (inmobi.com, jirbo.com, vungle.com, startapp.com, chartboost.com) belong to Advertising and Analytics business whereas others are different kinds of service providers (amazon.com, crashlytics.com, skyhookwireless.com). Here it is worth noting that `skyhookwireless.com` provides geolocation service among other kind of services. With this service, an application can get the location of the phone without explicitly requesting a geolocation related permission.

4.2 Dynamic analysis

We notice that 88 applications access, at least, two privacy-sensitive methods. We chose these applications for dynamic analysis to know what information they are sending over network and to which servers.

To perform this dynamic analysis, we use a modified system image of Android OS. Later, applications are run on a device flashed with this modified system image of OS. Our modified OS lets us track the behavior of a particular application. We modify the classes/methods of our interest and log interesting method calls in a local SQLite database. Going in more details, we modify some methods in `WifiManager` and `WifiInfo` classes along with network (both cleartext and SSL) and data modification (encryption and hash) related methods. This generated SQLite database is later analyzed with automatic scripts to know if a particular application is accessing some information and leaking it over network. However, it is worth mentioning that other parts of the OS remain untouched and the system is not different at all from Android in all other aspects.

As already explained in Section 3, applications could infer a lot of information about the user through the data available by this permission. To confirm these speculations, we test these 88 applications and check what information these applications (or third-party libraries included in them) have already started to send to remote servers. And to our surprise, yes, both first and third-parties have already started to do it.

Table 5 presents the list of servers to which privacy sensitive information obtained with

Table 5: Servers where Wi-Fi related information is sent by 88 dynamically analyzed applications.

Info	Third-parties	First-parties	# Apps
MAC Address	appsflyer.com(SSL), revmob.com(SSL), adsmogo.mobi(plain-text), adsmogo.org(plain-text), vungle.com(plain-text), supersonicads.com(plain-text), trademob.net(SSL), sponsorpay.com(SSL), beintoo.com(SSL), adsmogo.com(plain-text), 115.182.31.2/3/4(plain-text)[5], tapjoyads.com(SSL)	Not found	13
(B)SSID of connected AP	inmobi.com(SSL), 93.184.219.82(plain-text)	Not found	2
Wi-Fi Scan Info	inmobi.com(SSL), fastly.net(SSL)	badoo.com(SSL), foursquare.com(SSL)	5
Wi-Fi Config Info	Not found	Not found	0

the `ACCESS_WIFI_STATE` permission is sent. A number of third-parties present inside these applications are collecting Wi-Fi MAC Address and sending it to their servers (even in cleartext in some cases). Accessing Wi-Fi MAC address is really serious as it is a hardware-tied unique identifier that remains the same all along the lifespan of the device and can be used to tie both on-line and physical profile of a user (see Section 3.1). Looking at this list of servers in the Table 5 where Wi-Fi MAC address is sent, it seems like most of these servers belong to Advertising and Analytics companies. This is not a surprise since those actors need a unique identifier to track the users.

Also, both first (Badoo.com) and third-parties (inmobi.com) collect the SSID and BSSID of the AP to which the device is connected. Let's imagine them having this database of all users and the Wi-Fi APs to whom they connect to. It might easily reveal various relationships between users if two users connect to the same Wi-Fi AP. Depending on the type of Wi-Fi APs to which users are connected to (for example, a protected Wi-Fi at home/work or at what time/location they connect to), a lot of information could be derived about social links between users. As an illustration, they can know users sharing a common flat among 10 users with same geolocation. People living in a 10 story building might have approximately same geolocation retrieved by GPS but this information can help them to know which users really share a flat/apartment.

Moreover, there are applications who get list of surrounding Wi-Fi APs and send it to remote servers. For example, we found that Badoo and Foursquare applications sends the list of surrounding Wi-Fi APs (SSIDs, BSSIDs, signal strength, etc.) to their respective servers. Here it must be noted that both these applications have `ACCESS_FINE_LOCATION` permission and can get precise device geolocation by the Android system. But still they send the list of scanned nearby Wi-Fi APs to their servers. It leads to various speculations about the usage of this data. Do they sell this data to other parties? As these applications have wide user-base, the database built this way by these companies could be quite accurate; and there exists a possibilty that it can be sold to other companies for money.

Lastly, we even found some third-parties (for example, `inmobi.com`, `fastly.net`) sending

[5]Wi-Fi MAC Address is hashed (SHA-1) before sending over network in clear-text.

the list of surrounding Wi-Fi APs to their servers; they are present inside various applications. Focusing on the communication inside various applications to `inmobi.com` server, we find that `inmobi.com` library present inside these applications works in two modes: First, if it is included in an application having `ACCESS_FINE_LOCATION`, it accesses the fine-grained geolocation retrieved by the system along with nearby Wi-Fi APs (possibly to enrich their own database) and when an application doesn't have this permission, it derives device geolocation by querying their geolocation server with the list of surrounding Wi-Fi APs. As an example, code from `inmobi.com` inside SimSimi (com.ismaker.android.simsimi) application sends the list of surrounding Wi-Fi APs to its server to derive device geolocation because this application doesn't have neither `ACCESS_FINE_LOCATION` nor `ACCESS_COARSE_LOCATION` permissions.

Finally, we didn't encounter any application sending Wi-Fi configuration information over the network (which is good for our privacy) but this might be the case in near future. Also, it might be possible that our dynamic analysis technique couldn't detect the leakage of this information over network. In fact, one of the limitation of our technique is the fact that it misses the detection of data leakage if the data is modified by the application itself using their custom data modification methods instead of system methods (for example, methods or functions provided by Android to encrypt/hash data). In all cases, we must take proper action now before it is too late.

5 User perception

This section focuses on studying user perception of the `ACCESS_WIFI_STATE` permission. Section 3 and 4 have respectively demonstrated the potential privacy threats and the actual situation today on Google Play. In fact, as studied in [16], it's not always easy for users to know the kinds of privacy-sensitive data could be made available by a permission. Android permissions are often misunderstood by users. So we conduct an on-line survey involving 156 Android users to study their perception of comparative privacy risks associated with `ACCESS_WIFI_STATE` permission as well as regarding the privileges attributed to applications by this permission.

5.1 Survey description

User attitude towards privacy has been measured thanks to the Westin index [22], that is a set of three questions often used in privacy studies, for example in [16], to evaluate users privacy concerns and behavior on the Internet. Our survey has been implemented using Google Docs and spread through social media and multiple mailing-lists. Spreading the survey in this manner is not ideal (it clearly is a limitation of the study) due to possible sampling issues.

The survey is composed of 12 questions divided into 3 parts: the first part focuses on demographic information such as age, gender and professional category; the second part is concerning user attitude towards privacy as well as to know since how long the user is using the Android system; and finally, the third part is to evaluate the user perception of relative privacy risks associated with several permissions and to know in detail how well users understand the implications of the `ACCESS_WIFI_STATE` permission. The permission were presented using a screenshot of the permission's description (as shown to the user by the Android system) rather than the corresponding permissions' names.

The third part of the survey starts with a series of questions where the respondent must give an evaluation of the privacy risks associated with 5 selected Android permissions on a scale of 1 to 10. Along with `ACCESS_WIFI_STATE` permission, we selected `CHANGE_WIFI_STATE` and `ACCESS_NETWORK_STATE` permissions in 'Network Communications' group to understand how the user differentiates permissions belonging to the same group but giving access to different

kinds of network-related data. We also selected `ACCESS_FINE_LOCATION` that is the permission explicitly required by applications to get device geolocation. As device can also be geolocalized indirectly by applications making use of the `ACCESS_WIFI_STATE` permission, `ACCESS_FINE_LOCATION` permission is selected to know how users evaluate the privacy risk of `ACCESS_WIFI_STATE` permission as compared to it. Finally, the `READ_CONTACTS` permission is selected as a reference, because the name clearly signifies associated privacy risk.

Why comparison with `ACCESS_FINE_LOCATION` permission One might argue that the geolocation information obtained using Wi-Fi APs might not be as precise/accurate as geolocalization obtained by GPS making use of `ACCESS_FINE_LOCATION`. However, in practice, Wi-Fi based geolocation is quite accurate in cities [18]. A device could be tracked with a median positioning error of 13-40 meters [11] as opposed to its conservative estimate of around 100m or more. On the other hand, Blum et al. [9] reported mean location errors of 10-30 meters using the GPS module of smartphones. This means that in dense Wi-Fi environments such as cities, the geolocation information obtained from `ACCESS_WIFI_STATE` and `ACCESS_FINE_LOCATION` permissions are almost in the same order of accuracy. In addition, we must note that contrary to GPS, Wi-Fi based geolocation can be used indoors and even if a user turns the GPS off in order to save battery.

Table 6: Results of the last part of the survey about the fine understanding of the `ACCESS_WIFI_STATE` permission. Results for the correct answers are shown in green cells.

With `ACCESS_WIFI_STATE` permission and an Internet access, an application can ...	Responses		
	True	False	Don't know
✓ Check if the device is connected to the Internet through Wi-Fi	89.74%	5.77%	4.49%
✗ Turn the Wi-Fi on or off	6.41%	85.26%	8.33%
✗ Get the list of your contacts	6.41%	86.54%	7.05%
✓ Get the list of surrounding Wi-Fi networks	75.00%	12.18%	12.82%
✓ Get the list of configured Wi-Fi networks	65.38%	16.67%	17.95%
✗ Connect the device to a Wi-Fi network	21.79%	67.31%	10.90%
✓ Get the device location	48.08%	41.67%	10.26%
✓ Get one of the device unique identifiers	46.79%	17.31%	35.90%
✓ Get some of the previously visited locations (even before the App is installed)	35.90%	42.95%	21.15%

5.2 Results of the survey

In total, 190 users completed the survey from February 22 to 27, 2014. However, we discarded responses from 34 users who have never used the Android system. So the results and analysis presented below are based on the responses of 156 users who have, at least, some experience of using the Android system.

Concerning the professional category of the respondents, more than 70% of the respondents are 'Scientific or Technical', 'Software' or 'Telecomunications'. The age declared by the respondents was *lower than 21 years* in 18% of the case, *between 21 and 30 years* in 50%, and *between 31 and 40 years* for 21 %, leaving less than 10% above 40 years old. After computing the

Westin index [22] of the respondents, we found that 47% of them are categorized as *privacy fundamentalist*, 49% are *privacy pragmatist* and less than 4% are *privacy unconcerned*.

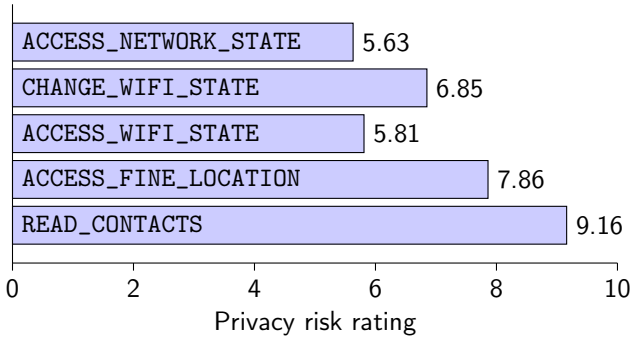


Figure 4: Averge privacy risk rating for considered permissions.

The responses for the set of questions related to the privacy risks ratings associated with each permission allowed us to have a comparative view of perceived privacy risks. Absolute average privacy risk ratings given by users on a scale of 1 to 10 is presented in Figure 4. Overall, `ACCESS_FINE_LOCATION` and `READ_CONTACTS` are rated the highest for privacy risks whereas `ACCESS_NETWORK_STATE` and `ACCESS_WIFI_STATE` are rated the lowest. In fact, users rate `ACCESS_WIFI_STATE` lower in terms of privacy risks than `ACCESS_FINE_LOCATION`. This means that users fail to perceive that device geolocalization could be derived from the information available to be accessed using `ACCESS_WIFI_STATE` permission. As not only device geolocalization but a lot of other PII could be obtained using `ACCESS_WIFI_STATE` permission as discussed in Section 3, it should have been, in fact, rated higher in terms of privacy risks than `ACCESS_FINE_LOCATION`.

The results corresponding to the last question about fine understanding of `ACCESS_WIFI_STATE` permission are presented in table 6. The correctness of the answers greatly vary across different questions. Thus we organized the questions asked into three groups based on the fraction of correct answers they received.

A first group of questions have been correctly answered by the majority of the respondents (more than 75% of correct answers). This first group includes questions about the basic functionalities of the `ACCESS_WIFI_STATE` permission (e.g., checking Internet connectivity through Wi-Fi and getting the list of surrounding Wi-Fi networks) as well as privileges that are not granted by the permission (e.g., turning the Wi-Fi on or off and getting the list of contacts).

The second group of questions have a lower rate of correct answers but there is still a majority (more than 60%) of respondents that gave the correct answer. It concerns the ability of the application to access the list of configured networks and the ability to connect the device to a Wi-Fi network.

Finally, the third group is the one that have received the lowest rate of correct answers (below 50%). Those questions concerns the ability of getting current or past geolocation information as well as one of the device unique identifier. We remark that these poorly understood capabilities are also the most privacy invasive. Even though a majority of the respondents failed to correctly answer the last set of questions, there is still a significant proportion of respondents (more than 35% correct answers) who answered it correctly.

Focusing on the geolocation capability of the `ACCESS_WIFI_STATE` permission, we took a closer look at the relative rating of the `ACCESS_FINE_LOCATION` compared to the `ACCESS_WIFI_STATE` permission. We only considered the respondents who have correctly answered that

device geolocation could be obtained using the `ACCESS_WIFI_STATE` permission. One would expect that, at least, these respondents would give the same rating to both `ACCESS_WIFI_STATE` and `ACCESS_FINE_LOCATION` permissions. However, we found that this group of respondents rate `ACCESS_WIFI_STATE` permission 2.55 points less risky in terms of privacy than the `ACCESS_FINE_LOCATION`. This means that despite a clear understanding of the geolocation capability, respondents tend to consider the `ACCESS_WIFI_STATE` less risky than the `ACCESS_FINE_LOCATION` permission.

6 Related work

Inference of private information by application on the Android system have been studied in [23]. This work focuses on the exploitation of *public* information available on the the Android system, i.e., data that do not require any permission to be accessed. In particular, they show that some PII (such as, geolocation, driving route, identity) can be inferred from the *public* information made available to be accessed for applications by the Android system. We note that getting device geolocation is common in both ours and this study. However, in [23], it can only be obtained when the device is connected to a Wi-Fi network, whereas in our work, it can be obtained as long as the Wi-Fi interface is enabled.

The problem of overprivileged applications has been studied in [15]. The study shows that a third of the applications are requesting more permissions than actually required. One of the proposed explanations is the confusion between permission names, which is typically the case for network related permission such as `ACCESS_WIFI_STATE` and `ACCESS_NETWORK_STATE`. Our results are in line with those findings as we have identified 17% of the applications requesting `ACCESS_WIFI_STATE` permission without really using it.

The user comprehension of Android permissions has been studied by Felt et al. in [16], using a questionnaire as we did. The authors considered a total of 11 permissions including `READ_PHONE_STATE` and `CHANGE_NETWORK_STATE`, but they did not consider the `ACCESS_WIFI_STATE` nor the `ACCESS_FINE_LOCATION` permissions. Like our study of the `ACCESS_WIFI_STATE` permission, [16] shows that many users have a poor understanding of the Android permission system.

7 Conclusion and Potential Solutions

The paper, first, presented what PII could be directly obtained or indirectly derived from data made available to be accessed for applications by the `ACCESS_WIFI_STATE` permission. We showed that a large number of applications request this permission and then, with the help of an online survey, we found that users often fail to perceive privacy implications associated with this permission. Our analysis of a representative set of most popular applications in each category on Google Play revealed that a number of both first and third-parties have already started to exploit this permission to access or derive user PII.

The results of this study call for changes in the Android permission system: first the access to Wi-Fi scan results should be protected with location permissions; then the `ACCESS_WIFI_STATE` permission description should indicate the various PII that can be directly obtained or inferred from it; finally, the `ACCESS_WIFI_STATE` permission should be placed in the list of dangerous permissions as it is more privacy-sensitive than some of the permissions already in the list.

References

[1] Androguard. `https://code.google.com/p/androguard/`.

[2] Android System Permissions Categories. `http://developer.android.com/guide/topics/manifest/permission-element.html`.

[3] Android System Permissions Groups. `http://developer.android.com/reference/android/Manifest.permission_group.html`.

[4] Google Play unofficial API. `https://github.com/egirault/googleplay-api`.

[5] The Google Maps Geolocation API. `https://developers.google.com/maps/documentation/business/geolocation/`.

[6] This recycling bin is following you. Quartz.com. 8 août 2013. `http://qz.com/112873/this-recycling-bin-is-following-you/`.

[7] WiFiManager Class. `http://developer.android.com/reference/android/net/wifi/WifiManager.html`.

[8] WiGLE: Wireless Geographic Logging Engine. `http://wigle.net/`.

[9] J. R. Blum, D. G. Greencorn, and J. R. Cooperstock. Smartphone sensor reliability for augmented reality applications. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 127–138. Springer, 2013.

[10] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal analysis of android ad library permissions. *arXiv preprint arXiv:1303.0857*, 2013.

[11] Y.-C. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm. Accuracy characterization for metropolitan-scale wi-fi localization. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, pages 233–245, New York, NY, USA, 2005. ACM.

[12] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, 1996.

[13] M. Cunche, M.-A. Kaafar, and R. Boreli. Linking wireless devices using information contained in wi-fi probe requests. *Pervasive and Mobile Computing*, (0):–, 2013.

[14] C. Daniel and W. Glenn. Snoopy: Distributed tracking and profiling framework. In *44Con 2012*, 2012.

[15] 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 '11, pages 627–638, New York, NY, USA, 2011. ACM.

[16] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[17] B. Greenstein, R. Gummadi, J. Pang, M. Y. Chen, T. Kohno, S. Seshan, and D. Wetherall. Can Ferris Bueller still have his day off? protecting privacy in the wireless era. In *Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 10:1–10:6, Berkeley, CA, USA, 2007. USENIX Association.

[18] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, et al. Place lab: Device positioning using radio beacons in the wild. In *Pervasive computing*, pages 116–133. Springer, 2005.

[19] J. Lindqvist, T. Aura, G. Danezis, T. Koponen, A. Myllyniemi, J. Mäki, and M. Roe. Privacy-preserving 802.11 access-point discovery. In *Proceedings of the second ACM conference on Wireless network security*, pages 123–130. ACM, 2009.

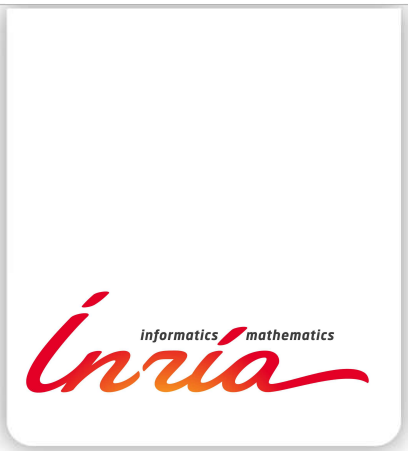
[20] X. Liu, S. Zhang, F. Wei, and M. Zhou. Recognizing named entities in tweets. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 359–367. Association for Computational Linguistics, 2011.

[21] A. B. M. Musa and J. Eriksson. Tracking unmodified smartphones using Wi-Fi monitors. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 281–294, New York, NY, USA, 2012. ACM.

[22] H. Taylor. Most people are 'privacy pragmatists' who, while concerned about privacy, will sometimes trade it off for other benefits. *The Harris Poll*, 17(19):44, 2003.

[23] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1017–1028, New York, NY, USA, 2013. ACM.

Contents



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399