Short Paper: WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission*

Jagdish Prasad Achara†
Mathieu Cunche‡† and Vincent Roca†
†Inria, Grenoble, France
‡University of Lyon, France
firstname.lastname@inria.fr

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

ABSTRACT

On Android, installing an application implies accepting the permissions it requests, and these permissions are then enforced at runtime. In this work, we focus on the privacy implications of the `ACCESS_WIFI_STATE` permission. For this purpose, we analyzed permissions of the 2700 most popular applications on Google Play and found that the `ACCESS_WIFI_STATE` permission is used by 41% of them. We then performed a static analysis of 998 applications requesting this permission and based on the results, chose 88 applications for dynamic analysis. Our analyses reveal that this permission is already used by some companies to collect user Personally Identifiable Information (PII). We also conducted an online survey to study users' perception of the privacy risks associated with this permission. This survey shows that users largely underestimate the privacy implications of this permission. As this permission is very common, most users are therefore potentially at risk.

Categories and Subject Descriptors

K.4.1 [**Public Policy Issues**]: Privacy

Keywords

Android Permissions, Wi-Fi, Personally Identifiable Information (PII) leakage, Static/Dynamic Analysis, User survey

1. INTRODUCTION

Mobile devices have become ubiquitous and are a crucial part of our lives today. These devices handle a lot of private data such as our email, communications, location, etc. As useful as they are, they also became a serious threat to user's privacy. Indeed, very accurate profiles can be created using the vast amount of data available on them. Advertising and Analytics (A&A) companies have therefore shifted their focus from traditional desktop computers and browsers to applications running on mobile devices.

Today a large fraction of mobile devices are running Android, where access to user data is controlled by the permission system. However, users have a poor understanding of the permissions and do not pay enough attention to these messages; they therefore often do not realize the kind of information accessible to an application once installed [7]. In order to help users perceive the potential risks, Android classifies the permissions based on their expected risk[1]. `ACCESS_WIFI_STATE`, on which we focus in this work, allows an application to access various information related to the the Wi-Fi interface. As such it is categorized as `normal` as compared to, e.g., `ACCESS_FINE_LOCATION` which is classified as `dangerous`. In this work we show that the `ACCESS_WIFI_STATE` permission is actually `dangerous` as a user's PII can be (and is already) derived from the use of this permission. Also, as the use of this permission by ad libraries has increased over time [3], the problem is severe.

More specifically the contributions of this work are threefold: First, we consolidate what PII can be derived from the data related to Wi-Fi interface (Section 3), namely unique identifiers (useful for tracking purposes), device geolocation, travel history and social links between users. Second, we analyze the current situation on Google Play employing both static and dynamic analysis of Android applications (Section 4), revealing that a large number of applications have actually started to exploit this permission to obtain user PII. 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 do not understand the privacy implications of this permission.

2. BACKGROUND AND RELATED WORK

2.1 Android Permission System

As Android gives no privilege to applications by default, applications must ask the user for privileges by statically declaring the list of permissions it requires. 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

*A full version of this paper is available at [1].

[1] `http://developer.android.com/guide/topics/manifest/permission-element.html`

WiSec'14, July 23–25, 2014, Oxford, UK.

http://dx.doi.org/10.1145/2627393.2627399.

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.	(Currently being) enabled/disabled or 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

the permissions required to respectively geolocalize the device and read user's contact data).

The `ACCESS_WIFI_STATE` permission. It falls in the '`Network communications`' group of permissions and makes some of the methods of the `WifiManager` class[2] available to be accessed by applications (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.

2.2 Related Work

Zhou et. al. [13] show how PII (e.g., geolocation, identity) can be inferred from *publicly* available information in the Android system. We note that getting device geolocation is common in both ours and [13]. However, in [13], device geolocation is only obtained when the device is connected to a Wi-Fi network, whereas in our study, we show that it can be obtained as long as the Wi-Fi interface is enabled.

Book et al. [3] investigates changes over time in the behavior of Android ad libraries. The study reveals that number of libraries able to use different permissions, `ACCESS_WIFI_STATE` permission is one among them, drastically increased over time. Also, Nguyen et al. [10] show that Wi-Fi information could be used to breach location privacy. However, [3] and [10] did not analyze the current situation on Google Play. As opposed to these studies, our study employs in-depth analysis of applications requesting `ACCESS_WIFI_STATE` permission and shows that a number of user PII can be and are already derived from the data accessible using this seemingly network-related permission.

The user comprehension of Android permissions have been studied in [7]. The study considered a total of 11 permissions but left out `ACCESS_WIFI_STATE` and the `ACCESS_FINE_LOCATION` permissions. Likewise our specific study of `ACCESS_WIFI_STATE` permission, the results of this study also indicate that users have a poor understanding of the Android permission system.

3. USER PII INFERRED FROM WI-FI 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 innocuous, but it is actually possible to either directly access or infer several user PII. In this section, we describe such user PII.

A unique device identifier. Using the `getConnectionInfo()` method, an application can obtain the MAC address of the Wi-Fi interface. In fact, there are other hardware-tied identifiers available to be accessed by applications, e.g., `IMEI` and `MEID` with `READ_PHONE_STATE` permission. As these unique identifiers could be used by advertisers to track user activities across all applications, they pose serious threat to user privacy. In particular, Wi-Fi MAC is also used to track users in the physical world [12, 6] and therefore, allows trackers to link both online and physical profiles of the user.

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. This method does not trigger the scanning of Wi-Fi APs, but return the list of last scanned Wi-Fi APs. Wi-Fi scan is performed automatically by the system every 15 seconds and can also be triggered by other applications, therefore this list of surrounding Wi-Fi APs is often up-to-date. By submitting the raw result of a Wi-Fi scan to a remote geolocation service[3], the device gets geolocation information (coordinates and accuracy metric) in return. Wi-Fi based geolocation is accurate to 20 meters in urban areas [9]. The Wi-Fi, GSM and GPS-based geolocation systems are employed by the Android system, but access to this information is restricted by the `ACCESS_FINE/COARSE_LOCATION` permissions. On the opposite, the raw Wi-Fi scan information is not protected by any of these two geolocation permissions. By using a third party Wi-Fi based geolocation service, it is therefore possible for an application to obtain geolocation information without having to explicitly ask for geolocation permissions as long as the application has both the `ACCESS_WIFI_STATE` and `INTERNET` permissions. And in Section 4, we show that this is often the case.

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.

Travel history. The list of Wi-Fi networks to which the device has been connected to in the past 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 used. It has been shown in [8] that the data stored in the *Configured Networks List* can be combined with external resources[4] to obtain information such as the previously visited locations.

Social links. It is possible to predict the existence of a social or professional link between the owners of devices by

[2] `http://developer.android.com/reference/android/net/wifi/WifiManager.html`

[3] `https://developers.google.com/maps/documentation/business/geolocation/`

[4] `http://wigle.net/`

Table 2: Most commonly used 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%)

comparing the list of SSIDs known by any two devices [5]. 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 *Configured Networks List* returned by the `getConfiguredNetworks()` method can be used for the same purpose. In fact, it also contains some other information about the configured Wi-Fi AP (e.g., allowed protocols, authentication algorithms, key management). This information can be leveraged to improve the quality of the social link establishment as [5] relies only on the SSID.

Other PII derived from SSIDs. SSIDs are often made to be meaningful to users and potentially contain information about the network owner or its users (e.g., name of institutions, individuals or locations [11]). Extraction of this information can be done manually or could be automated using techniques like Named Entity Recognition [4] and could then be used, for example, by advertisers to enrich the profile of the user to serve targeted ads.

4. ANDROID APPLICATIONS ANALYSIS

We have analyzed the 100 most popular free applications in each of the 27 categories present on Google Play, i.e., 2700 applications in total. We focused only on those that require both the `ACCESS_WIFI_STATE` and the `INTERNET` permissions (only 5 applications ask for the first permission but not the second one). To that purpose, we crawled Google Play[5] and collected the list of required permissions. Then we statically analyzed 998 applications requesting these permissions and based on the results, we chose 88 applications for an in depth, dynamic analysis. This section details our findings.

4.1 Static analysis

We used custom scripts (based on Androguard[6]) to statically analyze the 998 APK files corresponding to applications requiring both the `ACCESS_WIFI_STATE` and the `INTERNET` permissions.

4.1.1 *Use of the* `WifiManager` *class' methods*

We note that ∼17% (165) of the applications request `ACCESS_WIFI_STATE` permission but do not access any of its methods. Those over privileged applications present a privacy risk as later revisions of those applications will be able to use the protected methods by this permission.

Table 2 presents the number of applications calling the `WifiManager` class' methods. In fact, ∼76% (762) applications are accessing at least one of these three privacy-sensitive methods whereas ∼1% (11) of them are accessing

[5]Using `https://github.com/egirault/googleplay-api`
[6]`https://code.google.com/p/androguard/`

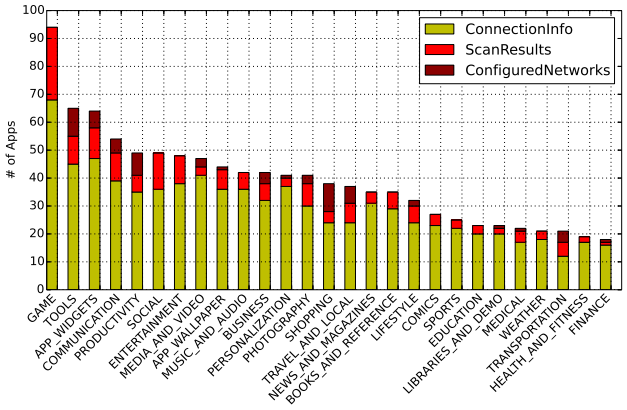

Figure 1: Per category popularity of 3 privacy-sensitive methods (100 applications/category).

all 3 privacy sensitive methods. Among the 6 methods protected by this permission, we chose to focus on the ones that pose serious privacy risks to the user, namely: `getScanResults()`, `getConfiguredNetworks()` and `getConnectionInfo()`. Note that the access to `WifiManager` class' methods might be legitimate, for example, in case of a Wi-Fi manager application in the `Tools` or `App Widget` categories, but probably not for a cooking or wallpaper application. From now on, our analysis only focuses on these three methods.

Figure 1 presents a per category distribution of the applications accessing these privacy-sensitive methods. Overall, applications in the `Game` category are the ones that use these 3 privacy-sensitive methods the most. There are also several other categories of applications that show a high usage of those methods without an obvious reason, like `Lifestyle`, `Comics` and `App Wallpaper`.

4.1.2 *Analysis of third-party code inside applications*

It is interesting to identify if the method calls are made by code written by the application developer ("first-party") or by the libraries (e.g., advertisement, analytics, performance monitors, crash reporters) included by the application developer ("third-party"). For this purpose we use a heuristic: classes belonging to a package whose name is the same as the application package name is considered coming from the application developer, otherwise it is considered as third-party code. Based on this, we find that 18% (136) of the applications accessing at least one of these methods are doing so only from third-party code. This confirms that third-party code is often responsible for the usage of the `ACCESS_WIFI_STATE` permission. Access to an API call by third-party code is sometimes legitimate, but there are cases where it is clearly not (e.g., Wi-Fi scanning is performed by `inmobi.com` and `skyhookwireless.com` in 13 applications out of 156 that scan for Wi-Fi APs).

Figure 2 presents, for each of the three privacy-sensitive methods, whether a first-party, third-party, or both access these methods. It reveals that there are some applications in which only the third-party code accesses these privacy-sensitive methods. This means that if the code written by the application developer does not require the `ACCESS_WIFI_STATE` permission, the third-party library does need it. The motivation for a third party library can be to secretly collect user information, or to provide a functionality

Table 3: Top 5 third-parties in each category and their corresponding number of applications.

ConnectionInfo		ScanResults		ConfiguredNetworks	
Third-party	# Apps	Third-party	# Apps	Third-party	# Apps
inmobi.com	74	inmobi.com	9	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

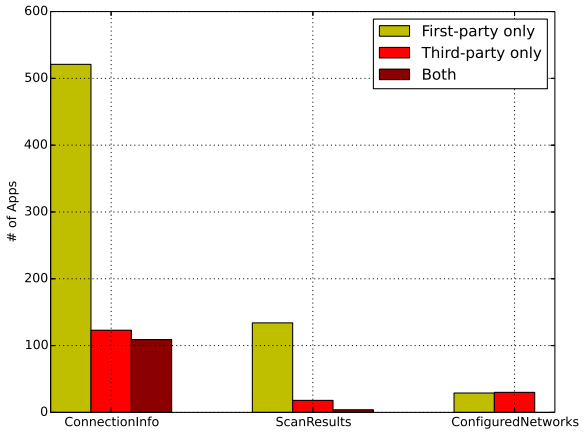

Figure 2: Distribution based on the party accessing privacy-sensitive methods, in 762 applications.

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 4 applications (Table 3). In any case deriving device geolocation without any explicit user permission is not legitimate and should be prevented by Android.

Table 3 presents the top 5 third-parties in each category and the number of applications in which they are present. Looking at the web pages 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`, `chartboost.com`) belong to A&A business, whereas others are different kinds of service providers, e.g., `skyhookwireless.com`. Here it is worth noting that `skyhookwireless.com` provides geolocation service among other kinds of services. With this service, an application can get the location of the phone with the use of `ACCESS_WIFI_STATE` and `INTERNET` permissions and without explicitly requesting a geolocation related permission.

4.2 Dynamic analysis

In Section 3 we speculated that applications can infer PII using the methods made available with the `ACCESS_WIFI_STATE` permission. We now analyze their network communications to check if these applications (or third-party libraries they embed) actually send private information to remote servers. We performed this dynamic analysis on 88 applications that access, at least, two privacy sensitive methods. For this purpose, we modified the Android OS to log interesting method calls in a local SQLite database. We modified several methods in `WifiManager` and `WifiInfo` classes along with network (both in clear and with SSL) and data modification (encryption and hash) related methods. The rest of the OS remains unmodified. This SQLite database is later analyzed to know if a particular application is accessing some information and leaks it over the network.

Table 4 presents the list of servers to which PII obtained with the `ACCESS_WIFI_STATE` permission was sent. A number of third-parties present inside these applications are collecting the Wi-Fi MAC address and send it to their servers (sometimes in clear). 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). Looking at this list of servers in the Table 4 where Wi-Fi MAC address is sent, most of them belong to A&A companies. This clearly suggests that those actors use the MAC Address as a unique identifier to track 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. Such a database of users and their Wi-Fi APs can easily reveal various relationships between users: a lot of information can be derived on the social relationships between users based on type of Wi-Fi APs to which users are connect to. For example, a protected Wi-Fi at home/work or the time/location at which two users connect to reveal close connections between them.

We found that Badoo and Foursquare applications send the list of surrounding Wi-Fi APs (SSIDs, BSSIDs, signal strength, etc.) to their respective servers. However, both applications have `ACCESS_FINE_LOCATION` permission and can get precise device geolocation from the regular Android APIs.

We even found some third-parties (e.g., `inmobi.com` and `fastly.net`) sending the list of surrounding Wi-Fi APs to their servers, and 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: 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); otherwise, if the 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 has neither the `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

[7]The Wi-Fi MAC address is hashed (SHA-1) before being sent over the network in clear-text.

Table 4: Servers where Wi-Fi related information is sent by 88 dynamically analyzed applications.

Info	Third-parties	First-parties	# Apps affected
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)[7], 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

for privacy) but this might be the case in near future. Also it might be possible that our dynamic analysis technique could not detect PII leakage in case applications employ custom data modification methods.

5. USER PERCEPTION

Sections 3 and 4 have respectively demonstrated the potential privacy threats and the actual situation today on Google Play. In this section we study how users perceive the `ACCESS_WIFI_STATE` permission. More precisely we conducted an on-line survey involving 156 Android users and we studied their perception of the privacy risks associated with this permission. We show that Android permissions are often misunderstood by users who do not necessarily understand their privacy implications [7].

5.1 Survey description

Our survey has been performed with Google Docs and diffused through social media and multiple mailing-lists. It was 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 about user attitude towards privacy and user's experience in using the Android system;
- the third part evaluates user's perception of the relative privacy risks associated with several permissions, and in particular how users understand the implications of the `ACCESS_WIFI_STATE` permission.[8]

The third part of the survey starts with a series of questions where the respondent must evaluate 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 the '`Network Communications`' group to understand how the user differentiates permissions belonging to the same group but giving access to different type of network-related data. We also selected `ACCESS_FINE_LOCATION` that is the permission explicitly required by applications to get device geolocation. As a device can also be geolocalized indirectly by applications having the `ACCESS_WIFI_STATE` permission, the `ACCESS_FINE_LOCATION` permission is selected in order to compare how users evaluate the privacy risks of both permissions. Finally, the `READ_CONTACTS` permission is selected as a reference since the name clearly indicates the associated privacy risks.

One might argue that the geolocation information obtained using Wi-Fi APs might not be as accurate as the geolocalization obtained through GPS with the `ACCESS_FINE_LOCATION` permission. However Wi-Fi based geolocation can be as accurate as GPS in urban scenarios [9, 2]. In addition, contrary to GPS, the Wi-Fi based geolocation can be used both indoors and when a user turns the GPS off to save battery.

5.2 Results of the survey

In total, 190 users completed the survey from February 22 to 27, 2014. We discarded responses from 34 users who never used an Android device. So the results and analysis presented below are based on the responses of 156 users who have some experience with Android.

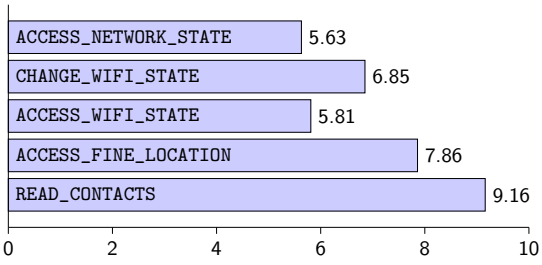

Figure 3: Average privacy risk rating for the considered permissions on a scale of 10.

The responses to the questions for each permission allowed us to have a comparative view of the perceived privacy risks. The average privacy risk ratings on a scale of 10 is presented in Figure 3. 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 particular, users rate `ACCESS_WIFI_STATE` as less risky than `ACCESS_FINE_LOCATION`. This is typically an error: not only geolocalization but also many other PII can be obtained through it (see Section 3). Therefore the privacy risks of this permission should have been rated higher than that of `ACCESS_FINE_LOCATION`.

The results for the question about the fine understanding of `ACCESS_WIFI_STATE` permission are presented in table 5. The correctness of the answers greatly varies across the questions. Thus we organized them into three groups, based on the fraction of correct answers they received.

The first group includes questions correctly answered by the majority of respondents (more than 75% of correct answers). We find questions about the basic functionalities

[8]The permission were presented using a screenshot of the permission's description (as showed to the user by the Android system).

Table 5: User understanding of the ACCESS_WIFI_STATE permission. Correct answers are shown in green cells.

With ACCESS_WIFI_STATE permission and an Internet access, an application can ...	Responses		
Options	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%

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 includes questions having received a majority of correct answers, but fewer than for the first group (in practice more than 60%). We find questions about the ability of the application to access the list of configured networks and to connect the device to a Wi-Fi network.

Finally, the third group includes questions having received the lowest rate of correct answers (below 50%). Those questions concerns the ability of getting current or past geolocation information as well as a 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 who answered correctly (more than 35% correct answers).

6. CONCLUSION AND POTENTIAL SOLUTIONS

The paper, first, presented what PII could be directly obtained or indirectly derived from data accessible to applications thanks to 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 of all, the access to Wi-Fi scan results should be protected with location permissions as is currently the case to access neighboring cell towers information. Secondly, the ACCESS_WIFI_STATE permission description should explicitly state the various PII that can be directly obtained (e.g., MAC address that can be used for tracking) or inferred from it (e.g., travel history). 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.

7. REFERENCES

[1] J. P. Achara, M. Cunche, V. Roca, and A. Francillon. WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission. INRIA Research Report N°8539, http://hal.inria.fr/hal-00994926/en, May 2014.

[2] 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*. Springer, 2013.

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

[4] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1), 1996.

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

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

[7] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. SOUPS '12, New York, NY, USA, 2012. ACM.

[8] 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 *USENIX HotOS workshop*, 2007.

[9] 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*. Springer, 2005.

[10] Y. T. Le Nguyen, S. Cho, W. Kwak, S. Parab, Y. Kim, P. Tague, and J. Zhang. Unlocin: Unauthorized location inference on smartphones without being caught.

[11] J. Lindqvist, T. Aura, G. Danezis, T. Koponen, A. Myllyniemi, J. Mäki, and M. Roe. Privacy-preserving 802.11 access-point discovery. In *ACM WiSec*, 2009.

[12] A. B. M. Musa and J. Eriksson. Tracking unmodified smartphones using Wi-Fi monitors. In *ACM SenSys'12*, 2012.

[13] 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 *ACM CCS 2013*.