UnLocIn: Unauthorized Location Inference on Smartphones without Being Caught

Le Nguyen, Yuan Tian, Sungho Cho, Wookjong Kwak,
Sanjay Parab, Yuseung Kim, Patrick Tague and Joy Zhang
Carnegie Mellon University
Moffett Field, CA, US
firstname.lastname@sv.cmu.edu

***Abstract*—Location privacy has become one of the critical issues in the smartphone era. Since users carry their phones everywhere and all the time, leaking users' location information can have dangerous implications. In this paper, we leverage the idea that Wi-Fi parameters not considered to be "sensitive" in the Android platform can be exploited to learn users' location. Though the idea of using Wi-Fi information to breach location privacy is not new, we extend the basic idea and show that clever attackers can do so without being detected by current malware detection techniques. To achieve this goal, we develop the Unauthorized Location Inference attack (UnLocIn) that is transparent to both the victim user and the malware detection software, using the seemingly insensitive permission to access Wi-Fi state. This permission is used by 51 of the top 100 free apps on Google Play. We demonstrate that the UnLocIn attack allows the attacker to infer the victim's location with 50 meter accuracy in 20% of cases and within a few hundred meters on average. In addition, we discuss potential defenses against our proposed UnLocIn attack.**

I. Introduction

Location privacy has become one of the hottest topics in the mobile era. Millions of apps are downloaded everyday, and many of them collect sensitive location data of mobile users. As shown in past research, modern smartphones are equipped with a variety of sensors that can be used to infer a user's location. In addition to GPS and cell tower trilateration [20], the embedded cameras [15] and microphones [7] can be considered as sources of sensitive location information.

Websites such as PleaseRobMe.com have made users aware of the dangers related to the potential misuse of location information. This awareness has caused an increasing number of users to pay more attention to the potential location privacy risks of mobile applications. Therefore, on the popular Android mobile platform, users are given control over their location privacy settings through a permission system. At installation time, a user can reject any application that requests suspicious access to sensitive information. Moreover, a user can explicitly deactivate location services such as GPS or install an application to fake the user's location.

Additionally, in order to protect users' location privacy, many researchers have proposed approaches for automatic detection and prevention of location information leakage. Conti et al. proposed a system to dynamically grant permissions based on the context of users [9]. For example, users can define location- and time-based constraints on the ability for applications to access GPS location. Dynamic information flow tracking is another widely used technique for detecting privacy violations. TaintDroid [11] is one such program that can be used for dynamic flow analysis in Android. It resides in the lower level of the Android OS and tracks the flow of sensitive data in third-party applications. TaintDroid will notify the user in the event that sensitive data leakage is detected.

While information flow tracking tools such as TaintDroid are a promising step toward providing user location privacy, we claim that it is an incomplete solution. To demonstrate our claim, we propose the Unauthorized Location Inference attack, which we refer to as UnLocIn, to exfiltrate sensitive location information without being detected by the user or leakage detection tools. Our goals in developing the UnLocIn attack rely on two primary capabilities: 1) to collect enough location data to localize the user/device and 2) to bypass current malware and data leakage detection methods. We developed and tested our UnLocIn attack to show that the two conditions are achievable on current Android devices. We achieve the first goal by exploiting the "access Wi-Fi state" permission, which is typically used to determine whether the phone is connected to the Internet through a Wi-Fi connection. According to our survey, 51 out of the 100 most popular free apps on Google Play such as Pandora, Angry Birds, or Skype ask for this permission. For our second goal, we analyze available malware detection methods and propose a generalized implicit information leak approach to bypass these methods. Given the exfiltrated data, we propose two techniques for inferring the user's location by exploiting publicly available location services. As part of our study, we perform various experiments to evaluate the accuracy of our location inference technique. Moreover, we discuss potential defenses against our proposed UnLocIn attack.

The remainder of this paper is organized as follows. In Section II, we detail the requirements for a successful attack and describe our proposed UnLocIn attack model. In Section III, we present methods for bypassing malware detectors. In Sections IV and V, we describe and evaluate the accuracy of location inference techniques for UnLocIn. Finally, we discuss potential defense mechanisms in Section VI and conclude the paper in Section VII.

II. UnLocIn Model and Requirements

The goal of the proposed UnLocIn attack is to track a user's location "without being caught". Thus, there are two primary requirements to make the UnLocIn attack successful:

- **Requirement 1**: The malicious application running on the victim's device should be *transparent to the user* and *transparent to malware detection tools*.
- **Requirement 2**: The malicious application should *collect and exfiltrate sufficient information* to infer the victim's location.

Together, these requirements ensure that the victim is neither directly aware of nor alerted by malware detection tools of the data collection process that enables the attack success. Note that we assume the malicious application is installed on the victim's device; this can be achieved by repackaging a popular application or developing malware masked as a game, each of which is out of the scope of this work.

In order to meet the two attack requirements, we design the UnLocIn attack with two parts, as shown in Figure 1. First, UnLocIn uses a malicious Android application installed on the victim's device to collect relevant data in background, occasionally reporting the collected data. Second, UnLocIn uses a server to collect and analyze the relevant data to infer the victim's location. As the UnLocIn application can be included in any standard Android application, it does not require the mobile device to be rooted.

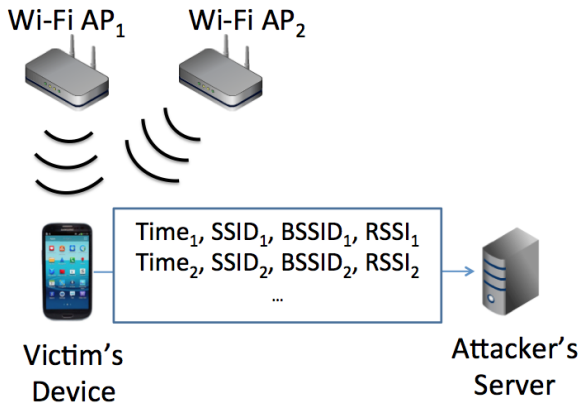

Fig. 1: In the UnLocIn attack, malware on the victim's device reports the SSID or BSSID of available Wi-Fi access points with timestapms to the attack server that performs location inference.

Common tools used to infer a user's location include GPS positioning and cell tower trilateration. However, if the UnLocIn malware tries to access these location sources directly, the victim will be notified at installation time about the required sensitive permission, as shown in Figure 2. In addition, the Android system automatically notifies the victim through an icon on the status bar during each request for GPS location access, as shown in Figure 3. Clearly the use of these location sources violates Requirement 1 for the UnLocIn attack. Therefore, an alternative data collection technique is required.

In order to meet Requirement 1 for the transparency of the UnLocIn attack, we exploit the seemingly insensitive Android permission to "access Wi-Fi state". This permission is typically used to check whether a user has a Wi-Fi connection and is able to transfer data at a high rate or in large quantity. However,

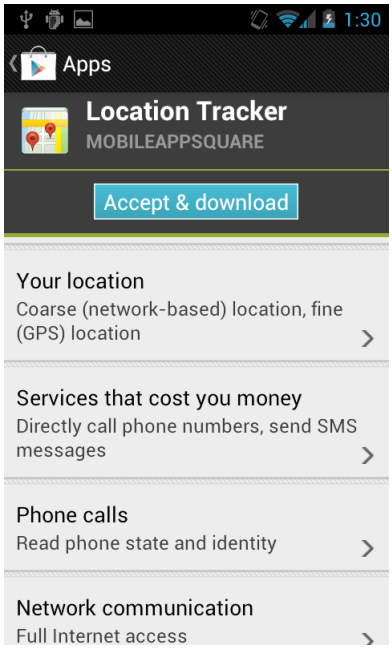

Fig. 2: The user is notified at installation time that the application requires location permissions.

Fig. 3: When GPS location is requested by an app, the Android OS notifies the user using a notification icon on the status bar (the icon on the left).

this permission also allows an application to obtain information about neighboring Wi-Fi Access Points (APs). At installation time, a victim will see only that the malicious application requests the "access Wi-Fi state" and "Internet connection" permissions. These permissions are common to most of the apps that require an Internet connection.

Once the malicious application is run for the first time, it periodically (e.g. every 5 minutes) scans for neighboring Wi-Fi APs. As a result it stores a list of SSIDs (Service Set Identification), BSSIDs (Basic Service Set Identification), RSSIs (Received Signal Strength Indication) with a corresponding timestamp. The application will then upload the data to the UnLocIn server, either through a periodic or opportunistic upload.

As we will describe later in Section IV, the information about surrounding Wi-Fi APs is sufficient for the location inference aspects of the UnLocIn attack, thereby fulfilling Requirement 1. In what follows, we will introduce our approach for bypassing malware scanners to perform the UnLocIn attack without being detected, thereby fulfilling Requirement 2.

III. Bypass Mechanism

In this section, we demonstrate how UnLocIn meets Requirement 1 for transparency to both user observation and malware detection. Our approach is based on an in-depth

investigation of both the Android permission system and the currently available malware detection tools. First, we only use non-sensitive permissions to bypass permission-based detection approaches. Second, for malware scanners which typically use dynamic flow analysis, we develop a technique to thwart their detection capabilities. More specifically, we use implicit information leakage and code obfuscation to avoid being caught. According to our experiments, the design goal is reached and our application successfully bypasses current detection techniques.

To bypass permission analysis tools, we only require permission to access Wi-Fi state, which is very commonly used. If permission analysis tools try to label it as a sensitive permission, many popular applications on the market will trigger a false alarm. According to our experiment, these popular permission-based analysis tools fail to detect our malicious behavior.

To bypass malware scanners which typically use dynamic flow analysis, we investigated various aspects of defenses against application-based attacks [16], [18], [8] and automated discovery of privacy leaks [21], [18], [10]. We found that tracking implicit leakage is a common limitation for dynamic flow analysis. For instance, implicit leakage shown in Algorithm 1 will not be detected by dynamic flow analysis.

Algorithm 1 Implicit Information Leak

```
location = get_gps_location()
if location == 1  then
   packet = 1
else if location == 2  then
   packet = 2
else if location == 3  then
   packet = 3
end if
send_network(packet)
```

However, if we only naïvely implement the algorithm for the implicit information leak, as shown in Algorithm 1, we would need to enumerate all possible locations through the if-else clauses. In our work, we design and implement a generalized implicit leak algorithm shown in Algorithm 2. In our approach, we break down the sensitive information into characters and then perform the implicit assignment. For example, to send the GPS latitude coordinate "-112.245", we break it down to "-", "1", "1", "2", ".", "2", "4", "5" and perform implicit assignment on the character level, thus avoiding enumeration of a large number of possible values.

Algorithm 2 Generalized Implicit Information Leak

```
unicode_hashmap = hashmap of unicode characters
bypassed_data = empty_string
for each character in sensitive_information
      find matching character in unicode_hashmap
            bypassed_data = bypassed_data +
                  character from unicode_hashmap
return bypassed_data
```

To evaluate our bypass approach of information flow analysis, we tested our app with TaintDroid [11], one of the most representative dynamic analysis tools to date. We first installed the UnLocIn Android application without any implicit routine on TaintDroid with Google Nexus S and Android 4.1.1. As soon as the application started, the TaintDroid system was able to catch the sensitive information leak, as shown in Figures 4 and 5. We then installed a modified version of UnLocIn using the implicit information leakage routines. The UnLocIn application ran over a few days, collecting the device's Wi-Fi scans and sending them to a remote server. The TaintDroid system was not able to detect any information leak during the whole testing period.

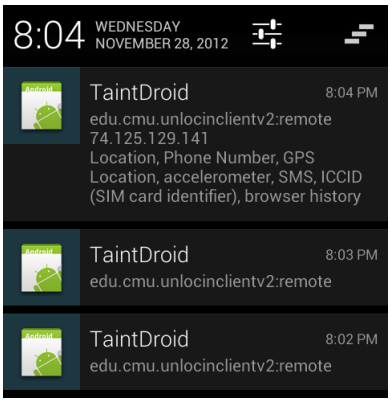

Fig. 4: TaintDroid Notifications

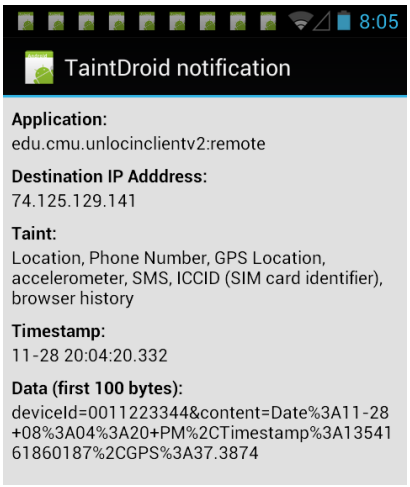

Fig. 5: TaintDroid Notification Details

From the above results, we learn that our app can bypass current analysis techniques by using limited permissions and the implicit information leak technique. In the current research, there has been a limited number of works addressing the implicit information leak, though static analysis can theoretically be used. Gilbert et al. proposed building a control dependencies graph to identify implicit information leakage [13]. However, this method runs into problems such as over-tainting, which results in the high rate of false positives [6]. Therefore, detection and prevention of implicit information leaks remains an open research problem, and our detection bypass mechanism allows us to satisfy Requirement 1 for the success of the UnLocIn attack.

IV. Location Inference

In this sections, we demonstrate how Requirement 2 can be satisfied by inferring a victim's location from Wi-Fi readings alone. Though the idea of using Wi-Fi for localization is not new, many approaches have been developed typically for controlled environments, for example using pre-calibrated systems in known environments to achieve high positioning accuracy. However, in our scenario, the victim's environment is completely unknown. In the following, we describe two location inference approaches based on BSSIDs and SSIDs collected from the victim's phone. Both proposed approaches use publicly available information sources and no pre-calibration is required.

A. BSSID-based Location Inference

1) Positioning based on Wi-Fi fingerprints: As described in Section II, the malicious application running on a victim's phone collects BSSIDs of observed Wi-Fi APs and their corresponding RSS values. It is broadly known that this information can be used for inferring the device's location.

In our attack, we need an access a Wi-Fi positioning database in order to infer the victim's location from the collected BSSIDs and RSS values. Google [1], Skyhook [4], Wigle [5] and other services have created a global Wi-Fi signature database by bootstrapping the database through war-driving [14]. Google and Skyhook allow developers to use their position inference service through restricted SDKs. Specifically on Android, the developer has an option to activate the positioning capability and expect to receive location information. The SDK will automatically request Wi-Fi information from the phone and return estimated location as a result. However, the SDK does not provide test functionality allowing a developer to simply input BSSID and RSS values to return a location, independent of a deployed app. Wigle exposes more capabilities to the developer community, but, based on our observations, Wigle has low geographical coverage and is not up-to-date in may areas. We are aware that Google and Skyhook provide their business partners a paid server-side location inference service. After signing a binding contract and paying a service fee, business partners get access to the functionality, which would allow them to input a set of BSSIDs and RSS values to get an inferred location. However, in an location tracking scenario, the attacker will likely want to remain anonymous, so these subscription services are not a viable option.

In our work, we decided to exploit the Google services for positioning purposes as their fingerprint database are well maintained and offer a good geographical coverage. Our goal is to implement a process where we can input BSSID and RSS and receive an output in form of latitude and longitude. We have achieved this by developing an location inference technique based on an Wi-Fi spoofing approach [19]. We use a Wi-Fi AP to broadcast the collected Wi-Fi fingerprints and use a Android phone to obtain the victim's location.

2) Tools: Figure 6 shows the attack tool set-up to infer a victim's location.

- **Android phone with Internet connection**: In our experiment we used the APIs of the Android SDK in order to infer the device's location. By putting the phone in airplane mode, we ensure that the phone does not use GPS or cell towers for positioning. However, in airplane mode, the Android phone allow us to keep Wi-Fi capabilities on. Thus, the device was able to receive the spoofed Wi-Fi beacon messages. Additionally, through a Wi-Fi connection, the device was connected to the Internet and was able to query Google's Wi-Fi fingerprinting database in order to infer the device's location.
- **Device capable of broadcasting Wi-Fi beacons messages**: In our experiment we used an external Wi-Fi access point in order to broadcast spoofed Wi-Fi beacons messages. However, any wireless internal card embedded in PCs or laptops can be used if they are supported by file2air, which is described below.
- **Faraday cage** (optional): As we conducted our experiment in an indoor environment, we used a Faraday cage in order to shield Wi-Fi signals from neighboring access points. Instead of building a Faraday cage the attacker could also find an environment with far from Wi-Fi access points.
- **File2air [2]**: We use file2air to send out spoofed Wi-Fi beacon frames.

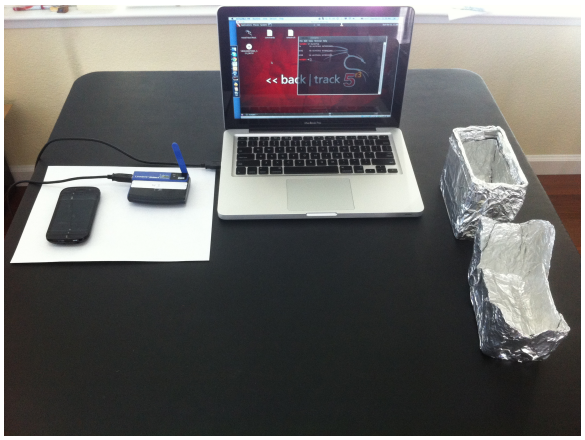

Fig. 6: We show our evaluation tools used for inferring the victim's location based on BSSIDs.

3) Location Inference: Figure 7 shows the process of location inference based on BSSIDs. As described in Section II, the malicious application running on the victim's phone collects the observed BSSIDs and sends them to the attacker. The attacker in a different location can use this information to infer the victim's location.

First, the attacker uses a Wi-Fi AP to broadcast Wi-Fi beacon packages with spoofed BSSIDs. Then the attacker places an Android phone in a vicinity to the Wi-Fi AP. An application running on the phone utilizes Google's positioning capability and stores the estimated location. Since GPS and cell tower sensors are deactivated in airplane mode, only Wi-Fi is used for estimating the device location. By broadcasting spoofed Wi-Fi beacons packets, the application queries Google's Wi-Fi fingerprint database and returns the location of the victim.

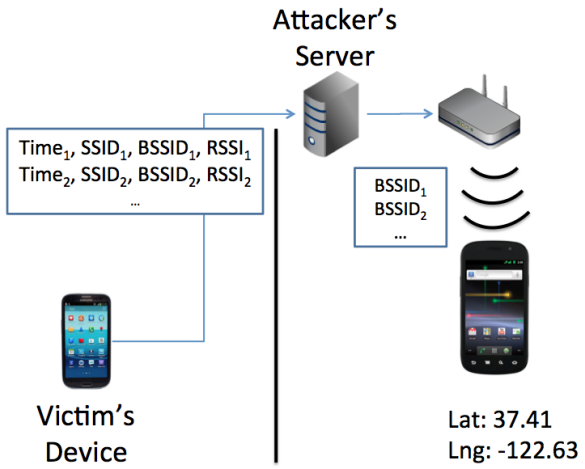

Fig. 7: We illustrate the process of inferring the victim's location through our replay approach.

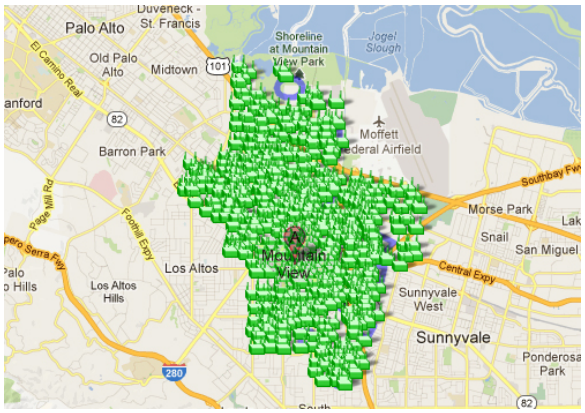

Fig. 8: Google's Wi-Fi coverage is illustrated.

Time	SSID	BSSID	RSSI
14:28:11	MSFTWLAN	...:23:c0	-85
14:28:11	MSFTWLAN	...:a7:80	-86
14:28:11	**MSFTGUEST**	...:23:c1	-86
14:28:11	MSFTGUEST	...:a7:81	-87
...	...	...	...
14:28:47	GoogleWiFi	...:85:c4	-76
14:28:47	GoogleWiFiSecure	...:85:c4	-76
14:28:47	GoogleWiFi	...:49:e8	-98
14:28:47	**CHM Public**	...:a7:82	-86
14:28:52	GoogleWiFiSecure	...:23:c0	-85
14:28:52	chmoffice	...:a7:80	-86
14:28:52	GoogleWiFi	...:23:c1	-86
14:28:52	GoogleWiFiSecure	...:a7:81	-87
14:28:52	GoogleWiFi	...:a7:82	-86
14:28:52	CHM Public	...:23:c0	-85

TABLE I: Keywords such as "guest" or "public" provide valuable search cues.

Time	SSID	BSSID	RSSI
21:05:16	**HolidayInn -Express/Santa Clara**	...:a7:82	-86
21:05:16	Metro_WiFi	...:23:c0	-85
21:05:25	MobileOne	...:23:c1	-86
21:05:25	HolidayInn -Express/Santa Clara	...:a7:81	-87
21:05:25	Metro_WiFi	...:85:c4	-76
21:05:25	Pinkberry	...:85:c4	-76
21:05:31	Capri Motel 002	...:23:c0	-85
21:10:15	**Sunnyvale Carwash**	...:23:c0	-87

TABLE II: SSID names can be informative to the location inference process.

B. *SSID-based Location Inference*

As mentioned above, in order to infer the victim's location from BSSIDs we assume that the utilized Wi-Fi fingerprint database has a sufficient coverage and contains fingerprints geographically close to victim's location. However, we cannot assume that these databases will cover all locations on the globe. Therefore, in some cases, the BSSID-based approach may not return a result. In what follows, we thus introduce an approach which uses SSIDs (instead of BSSIDs) to infer the victim's location.

One of the key insights of our work is that the SSIDs of Wi-Fi APs often contain sufficient information to leak a victim's location. Many of the public AP's SSIDs contain names of point of interests. Thus, even without training data, which is required by traditional Wi-Fi positioning methods, an attacker can infer a victim's location with low cost processing.

1) Positioning with Cues: We find that SSIDs associated with public or business places are good resources for location inference. Tables I and II show examples collected from a user's device. SSIDs such as "GoogleWiFi" and "GoogleWiFiSecure" are good examples. Since it is well known that Google provides free wireless Internet service in the city of Mountain View, CA [3], the attacker can narrow down the search space.

Next, cues that the attacker can exploit are keywords such as "public" or "guest", as shown in Table I. "CHM Public" or "MSFTGUEST" are candidates the attacker might be interested in. MSFT is the NASDAQ stock quote of Microsoft Corporation and its office is located right next to Computer History Museum (CHM) in Mountain View, CA. In many cases, as shown in Table II, SSIDs carry names of commercial stores (e.g. "Sunnyvale Carwash") and include geographical information (e.g. "HolidayInnExpress/Santa Clara").

In our work, we developed an approach to infer locations from the collected SSIDs. First, we applied pre-processing techniques such as eliminating SSID duplicates, replacing special characters with spaces, separating concatenated words and removing keywords such as "guest" and "public". Then we query online business/place directories in order to find the point of interests. Yelp and Foursquare are two publicly available APIs used in our experiments. In case a place is found based on pre-processed keywords, the APIs return a latitude and longitude of the place. We use this information to estimate the approximate location of a victim.

2) User Profiling with Timestamps and SSIDs: A combination of timestamps and SSIDs exposes additional sensitive information of a victim. For example, by considering timestamps an attacker can reconstruct the victim's moving trajectory and the transportation means. Figure 9 shows a map with exposed locations sorted by timestamps. Additionally, the map shows the inferred traveled trajectory. Since a victim moved from point A to point B in one minute, we can infer that a victim

drove a car (or was in a bus) on Bayshore Fwy.

Additionally, from the collected dataset we observed a significant correlation between an occurrence of a certain SSID and time of the day. Thus, by applying techniques proposed by Nguyen et al. [17], an attacker can infer where a victim works or where a victim typically goes for lunch. Additionally, the victim's future locations can be predicted based on the collected dataset.

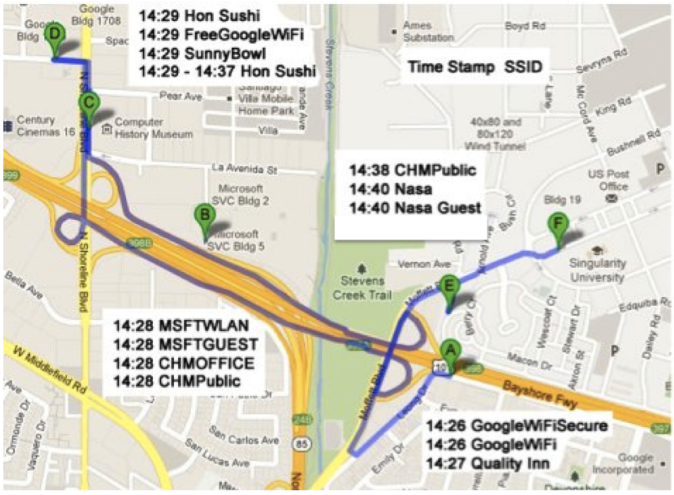

Fig. 9: Timestamps can assist in location tracking using SSIDs alone.

V. Evaluation

In order to evaluate the accuracy of the location inference technique proposed for use in the UnLocIn attack, we deployed our malicious application on a user's phone to simulate the attack. The phone was carried by a typical university student for one week. The application ran in the background and collected Wi-Fi readings and GPS locations (for ground truth in evaluating accuracy) every 5 minutes on average. In total, data from 1370 Wi-Fi scans was collected, containing 356 unique SSIDs and 498 unique BSSIDs.

A. Evaluating BSSID-based Location Inference

From the collected dataset, we observed that the person is in a stationary mode most of the time, i.e., the set of observed Wi-Fi BSSIDs remain similar. Thus, from the dataset we first identify all the unique sets of BSSIDs and infer the location only for them. For example, if at two different times t_1 and t_2, we observe the same set of BSSIDs, then we only need to infer the location at one of the times.

Due to the limitations of the used Wi-Fi AP, we were able to broadcast at most 4 concurrent Wi-Fi beacon frames. Thus, each of the 1380 Wi-Fi scans in our dataset corresponds to at most 4 BSSIDs. For each unprocessed set of BSSIDs, we thus picked the 4 with the highest RSSI values, in order to keep the most significant BSSIDs. From the original 1380 Wi-Fi scans, we obtained 470 unique sets of at most 4 BSSIDs.

We used a Wi-Fi AP to broadcast each of these 470 unique sets of BSSIDs. The application running on our Android device recorded both the observed BSSIDs and the locations predicted by the Android API. By putting the phone in the airplane mode and activating only the Wi-Fi capability we ensured that the phone used only Wi-Fi readings for the positioning purposes.

Through this process we were able to build a mapping between the unique set of BSSIDs and the corresponding location. We used this mapping to infer the location of the 1370 Wi-Fi scans. We were able to infer locations for 1136 of the total 1370 Wi-Fi scans (83%). For the remaining 17% of Wi-Fi scans the Android API did not returned any results. Through the analysis, we found that the victim was at a location where Google had not yet collected Wi-Fi fingerprints.

In order to evaluate the positioning accuracy, we collected the GPS positions of the victim's phone and used them as the reference positions. The *distance error* is computed as a distance between the position inferred through our approach and the reference position. Figure 10 shows the cumulative number of Wi-Fi scans having a distance error lower than a certain value. As we observe from the figure, in more than one third (513 of 1370) of the cases, we were able to predict the location with a distance error less than 250 meters, and in 20% of the cases (300 of 1370), the error was less than 50 meters. The average distance error for the 1136 Wi-Fi scans was less than 500 meters.

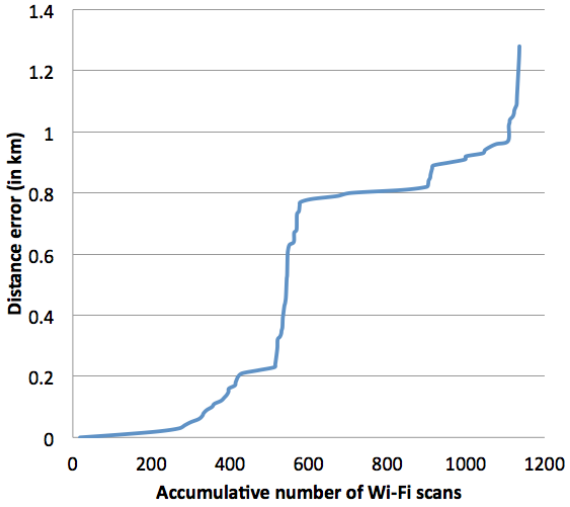

Fig. 10: We show the cumulative number of Wi-Fi scans having a distance error less than a certain value. For one third (513 of 1370) of the Wi-Fi scans, the distance error was less than 250 meters, and in 20% of the cases (300 of 1370), the distance error was less than 50 meters.

B. Evaluating SSID-based Location Inference

As shown in the previous section, we can successfully infer the victim's location based on the collected BSSIDs. This is possible since Google's Wi-Fi fingerprint database contains entries close to the location of the victim. However, as we observed, this fingerprint database does not have 100% geographical coverage, as there are cases in which the BSSID-based location inference ap proach does not return any results. In such cases, we can instead use the collected SSIDs to infer the victim's location.

We applied pre-processing of SSIDs as described in Section IV-B. Keywords obtained through pre-processing were input into the Foursquare and Yelp APIs. As the results, 18 SSIDs out of 356 unique SSIDs were matched to a certain location. Table III shows an example of one search result. We manually inspected the results and observed that all 18 SSIDs belong to known public places such as schools, universities, city libraries, restaurants, or companies. This result indicates that inferring a victim's location using SSIDs is practical, given that the set of SSIDs contains enough metadata such as the name of the business or other places.

SSID	**Sunnyvale Carwash**
GPS reference	37.353576, -122.013802
Foursquare	37.35323131084442, -122.013624
Yelp	37.3531021715992, -122.013816833496
distance(Foursquare)	41.4774487581538 meters
distance(Yelp)	52.7623597913769 meters

TABLE III: We illustrate an example search result using the Foursquare and Yelp APIs.

Along with the location search using Foursquare and Yelp APIs, we also analyzed the relative frequency of SSIDs to the particular victim. Table IV shows the top 7 BSSIDs with the number of Wi-Fi scans they appeared in. As we can see, the SSID "2WIRE964" appeared 682 times, about 50% of all the 1370 scans. By correlating the appearance of this SSID with the time of the day and the day of the week we were able to infer that this SSID is broadcast by a Wi-Fi AP near the victim's home. By observing locations of informative SSIDs that are accessed at similar times as "2WIRE964", an attacker can infer the approximate home location of the victim.

As the result of this experiment shows, SSIDs collected from a victim's device can leak a significant amount of sensitive information. We believe that this data leak have dangerous implications for a victim, similar to the risks publicized by websites such as PleaseRobMe.com.

Rank	SSID	Count
1	2WIRE964	682
2	DGI	679
3	mccoy	671
4	DesiNetwork	581
5	bones	571
6	CMU	455
7	CMU-SECURE	468

TABLE IV: We illustrate SSID inference results from our experiment.

VI. Defense Mechanism

To defend against the proposed UnLocIn attack, we describe two lightweight defense mechanisms based on fine-grained permissions and permissions assisted by static analysis.

A. Fine-grained Permission System

In this paper, we have shown how to exploit the Android permission system to infer users' location. This is possible through exploiting the seemingly insensitive permission "access Wi-Fi state". Through this permission, we were able to scan for surrounding Wi-Fi APs and collect sensitive information.

In order to address this issue, we propose the use of a more fine-grained permission system. From our survey, 51 out of 100 top apps on Google Play store use the permission "access Wi-Fi state". Most of these apps are bandwidth-sensitive apps (such as music streaming Pandora app or video calling Skype app). Thus, these apps use the "access Wi-Fi state" permission to detect stable Wi-Fi connection in order to optimize their data transfer. However, it is not clear how would they benefit from getting information about neighboring Wi-Fi APs. Therefore, we propose to extract the capability of reading information about Wi-Fi APs from "access Wi-Fi state" permission into a new "get information about Wi-Fi APs" permission. This new permission should be considered as sensitive as the permission for accessing coarse or fine-grained location information. Furthermore, we can extend the method to the whole permission system, i.e. dividing generic permissions into sensitive and non-sensitive parts.

B. Permission Analysis Assisted by Static Analysis

The proposed fine-grained permission system works well with existing permissions and resources. However, there may be new attacks utilizing new resources to infer sensitive information. This threat makes it necessary to apply a dynamically updated permission system. In order to detect new attacks, we can use feedback from static analysis tools to redefine the rules about dangerous permissions.

Figure 11 shows how to use feedback from static analysis to improve permission analysis. We maintain a dynamic sensitive permission database and use static analysis to update the database. If we identify malicious behavior from a permission which was labeled as non-sensitive, we will add this permission into sensitive permission database and vice versa.

In the initialization stage, we will deploy permission analysis, defining malicious sets of dangerous permissions to build a sensitive permission database, such as "read contact information" and "access internet" [12]. In the next step, we extract permissions required by the app and analyze them according to the sensitive permission database. As results from permission analysis might run into false negative and false positive to some extent, we propose an algorithm to update sensitive permission database for better accuracy by applying static analysis. For the case of false negatives, malicious behaviors are discovered by static analysis from an app with benign permission sets according to the permission analysis. In such cases, we will add permission sets considered as benign into the sensitive permission database. By this approach, we are enabled to discover new emerging attacks. On the other hand, if apps with some sensitive permission sets are actually benign from result of static analysis, those permission sets might not be used by real-world attackers very often. In such cases, there is a good reason to remove it from the sensitive permission database.

VII. Conclusion

We have demonstrated our design of the UnLocIn attack, performing inference of users' location using Wi-Fi state information in Android devices while remaining transparent to both user observation and malware detection tools. We have shown that UnLocIn can bypass current detection tools

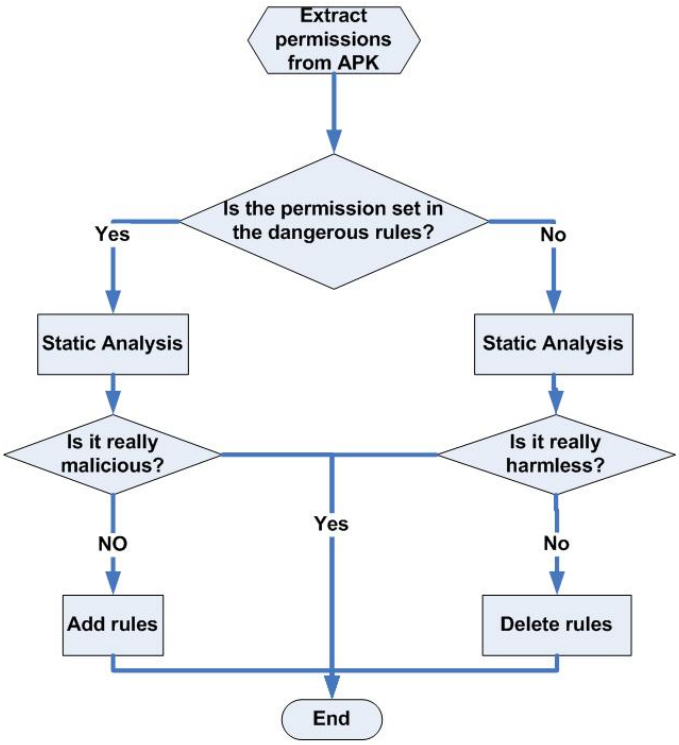

Fig. 11: We show how permission analysis can be assisted by static analysis.

and achieve a reasonable location accuracy to successfully track mobile users. In addition, we showed that both Wi-Fi BSSIDs and SSIDs can expose sufficient information to allow location inference using either APIs provided by location service providers or by mining SSIDs for names or locations that expose enough information on their own. Specifically, UnLocIn can track users to an accuracy of a few hundred meters, with the top 20% of cases achieving an accuracy of less than 50 meters. Finally, we discussed potential defense mechanisms against this type of attack using fine-grained permissions or a static analysis-assisted permission system.

REFERENCES

[1] Android sdk. http://developer.android.com.

[2] File2air. http://www.willhackforsushi.com/File2air.html.

[3] Googlewifi coverage. http://wifi.google.com.

[4] Skyhook. http://www.skyhookwireless.com.

[5] Wigle. http://wigle.net.

[6] L. Cavallaro, P. Saxena, and R. Sekar. On the limits of information flow techniques for malware analysis and containment. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 143–163, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] H.-T. Cheng, F.-T. Sun, S. Buthpitiya, and M. Griss. Sensorchestra: Collaborative sensing for symbolic location recognition. In M. Gris and G. Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 195–210. Springer Berlin Heidelberg, 2012.

[8] J. Clause, W. Li, and A. Orso. Dytan: A generic dynamic taint analysis framework. In *Proceedings of the 2007 international symposium on Software testing and analysis*, ISSTA '07, pages 196–206, New York, NY, USA, 2007. ACM.

[9] M. Conti, V. Nguyen, and B. Crispo. Crepe: Context-related policy enforcement for android. *Information Security*, pages 331–345, 2011.

[10] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song. Dynamic spyware analysis. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 18:1–18:14, Berkeley, CA, USA, 2007. USENIX Association.

[11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[12] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.

[13] P. Gilbert, B. Chun, L. Cox, and J. Jung. Vision: automated security validation of mobile apps at app markets. In *Proceedings of the second international workshop on Mobile cloud computing and services*, pages 21–26. ACM, 2011.

[14] M. Kim, J. J. Fielding, and D. Kotz. Risks of using ap locations discovered through war driving. In *Lecture Notes in Computer Science, v 3968 LNCS, Pervasive Computing - 4th International Conference, PERVASIVE 2006, Proceedings*, pages 67–82. Springer-Verlag, 2006.

[15] A. Mulloni, D. Wagner, I. Barakonyi, and D. Schmalstieg. Indoor positioning and navigation with camera phones. *IEEE Pervasive Computing*, 8(2):22–31, Apr. 2009.

[16] J. Newsome. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

[17] L. T. Nguyen, H.-T. Cheng, P. Wu, S. Buthpitiya, and Y. Zhang. Pnlum: system for prediction of next location for users with mobility. In *Proceedings of mobile data challenge by Nokia workshop at the tenth international conference on pervasive computing*.

[18] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 135–148, Washington, DC, USA, 2006. IEEE Computer Society.

[19] N. O. Tippenhauer, K. B. Rasmussen, C. Pöpper, and S. Čapkun. Attacks on public wlan-based positioning systems. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 29–40, New York, NY, USA, 2009. ACM.

[20] P. A. Zandbergen. Accuracy of iphone locations: A comparison of assisted gps, wifi and cellular positioning. *Transactions in GIS*, 13:5–25, 2009.

[21] Y. Zhu, J. Jung, D. Song, T. Kohno, and D. Wetherall. Privacy scope: A precise information flow tracking system for finding application leaks. Technical Report UCB/EECS-2009-145, EECS Department, University of California, Berkeley, Oct 2009.