The Perils of Wi-Fi Spoofing Attack Via Geolocation API and Its Defense

Xiao Han, Junjie Xiong, Wenbo Shen, Mingkui Wei, Shangqing Zhao, Zhuo Lu, and Yao Liu

***Abstract*—Location spoofing attack deceiving a Wi-Fi positioning system has been studied for over a decade. However, it has been challenging to construct a practical spoofing attack in urban areas with dense coverage of legitimate Wi-Fi APs. This paper identifies the vulnerability of the Google Geolocation API, which returns the location of a mobile device based on the information of the Wi-Fi access points that the device can detect. We show that this vulnerability can be exploited by the attacker to reveal the black-box localization algorithms adopted by the Google Wi-Fi positioning system and easily launch the location spoofing attack in dense urban areas with a high success rate. Furthermore, we find that this vulnerability can also lead to severe consequences that hurt user privacy, including the leakage of sensitive information like precise locations, daily activities, and demographics. Ultimately, we discuss the potential countermeasures that may be used to mitigate this vulnerability and location spoofing attack.**

***Index Terms*—Geolocation APIs, localization attacks, mobile and wireless security, Wi-Fi localization.**

I. Introduction

Wi-FI positioning systems have been commercialized and widely used as a complement to conventional GPS. Specifically, a mobile device relies on its 802.11a/b/g compatible wireless interface to collect Wi-Fi information, e.g., Medium Access Control (MAC) addresses about Wi-Fi access points (APs) in its vicinity. The mobile device sends this information to a Wi-Fi positioning system, which then looks up a database table that maps collections of Wi-Fi information to geographic locations and replies to the mobile device with a corresponding location.

Spoofing attacks against Wi-Fi positioning systems have been studied for over a decade. The basic idea is simple and straightforward. In particular, when at location B, an attacker simply broadcasts the Wi-Fi information collected from location A. Consequently, a mobile device at location B is deceived and obtains a wrong position estimate of location A from the Wi-Fi positioning system. Although this type of spoofing attack is easy to implement and seems to be effective, surprisingly, past research and practice show that such an attack can trivially impact a Wi-Fi positioning system, especially in urban environments with dense coverage of Wi-Fi APs. For example, the SkyLift, a low-cost Wi-Fi device that spoofs locations by using Wi-Fi microchip ESP8266, "may have little or no ability to spoof locations in dense urban environments with dozens of Wi-Fi networks" [19], and [47] mentions that spoofing attacks failed to spoof a device from its current location to a location far away, where the current location is covered by multiple public Wi-Fi APs.

Traditional attacks assume that the victim device is located in environments surrounded by few visible Wi-Fi APs (i.e., less than 5). In practice, however, a victim device normally receives Wi-Fi information from both the attacker and dozens of legitimate APs nearby, and reports all collected Wi-Fi information to the Wi-Fi positioning system. This implies that the existence of legitimate APs may interfere with the attacker's fake information in the decision-making process at the Wi-Fi positioning system. Alternatively, Tippenhauer et al. [47] proposes to eliminate Wi-Fi signals from legitimate Wi-Fi APs by conducting physical-layer jamming, such that the spoofing attack can achieve a better success rate. Nevertheless, jamming attacks require additional wireless equipment that is programmable at the physical layer. This can further complicate the practical implementation of the attack. Moreover, the victim device may be aware of jamming signals, and detecting the presence of jamming attacks has been extensively studied in the literature [56].

In this paper, we aim to address the following research questions. First, is it possible to spoof a Wi-Fi positioning system in dense urban areas without physical-layer jamming? If yes, what are the critical conditions that an attacker must meet? The main research challenge that we face to answer both questions is that location estimation algorithms that are adopted by a typical Wi-Fi positioning system are not public information. In other words, it is unclear how a Wi-Fi positioning system determines the position estimate when it receives Wi-Fi information from different locations. Using the example of Google, assume that the Google Wi-Fi positioning system receives mixed Wi-Fi information collected from Hamilton Park in New Jersey and Empire State Building in Manhattan in the same request. How does Google determine the location? Where is the position estimate result from Google?

Manuscript received 27 February 2023; revised 15 October 2023; accepted 18 December 2023. Date of publication 11 January 2024; date of current version 4 September 2024. *(Corresponding author: Xiao Han.)*

Xiao Han, Junjie Xiong, and Yao Liu are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: xiaoh@usf.edu; junjiexiong@usf.edu; yliu@cse.usf.edu).

Wenbo Shen is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: shenwenbo@zju.edu.cn).

Mingkui Wei is with the Department of Cybersecurity Engineering, George Mason University, Fairfax, VA 22030 USA (e-mail: mwei2@gmu.edu).

Shangqing Zhao is with the School of Computer Science, University of Oklahoma, Tulsa, OK 74135 USA (e-mail: shangqing@ou.edu).

Zhuo Lu is with the Department of Electrical Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: zhuolu@usf.edu).

Digital Object Identifier 10.1109/TDSC.2024.3352981

We aim to resolve this challenge and unveil the key factors that can significantly affect the success of the spoofing attack. We focus on the Google Wi-Fi positioning system, because it is the mainstream and most widely adopted system that supports the navigation and localization needs of over 1 billion people worldwide [16]. We find that Google provides the Geolocation API that enables a mobile device to obtain its location estimate by submitting collected Wi-Fi information to Google. Upon the request from this API, Google searches a location look-up table (LLT), which contains the locations of Wi-Fi APs in the connected world.

We find that an attacker can exploit the Geolocation API to reveal the location data residing in LLT at a negligible cost. We refer to this attack as *LLT inference attack*. Specifically, assume that the attacker detects multiple APs from the geographic location that he is currently at. Without the LLT inference attack, the attacker can only know that these APs are close to his current location. Nevertheless, by launching the LLT inference attack, the attacker can further identify the precise locations recorded by LLT for these APs. Knowing this information is important for the attacker because it can enable the attacker to launch successful spoofing attacks in dense urban environments without jamming. In particular, given the revealed locations of Wi-Fi APs in LLT, an attacker is capable of identifying the black-box localization algorithms used by the Google Wi-Fi positioning system. We discover four criteria enforced by Google in determining the position estimate when it receives mixed Wi-Fi information from multiple locations. These criteria can guide an attacker to intelligently craft malicious Wi-Fi information that deceives a victim device. The contributions of this paper are summarized below.

- We create the LLT inference attack that is capable of stealing the location data from the Google location database by exploiting the Google Geolocation API. As this attack enables one to obtain the precise locations of APs in LLT, we discuss that it may raise severe privacy concerns, such as inferring the relocation or travel of the user who carries and uses the AP. Besides the Google location service, we further study and identify the vulnerability of other location services (e.g., Mozilla).
- We demonstrate that an attacker can figure out the black-box localization algorithms used by Google based on the precise locations of APs revealed by the LLT inference attack. Specifically, the attacker can know how Google generates a position estimate in response to an API request that includes mixed Wi-Fi information from multiple locations.
- We further show that an attacker can construct a highly efficient location spoofing attack against the Google Wi-Fi positioning system with the knowledge of the precise locations of APs and the localization algorithms used by Google. Unlike the traditional spoofing attack, the discovered attack is more lightweight and stealthy, because it does not need to jam the Wi-Fi signals from legitimate APs for a good success rate in most cases.
- We discuss the potential countermeasures to mitigate the discovered location spoofing attacks. Our defense mechanism enables the victim device to differentiate legitimate APs from falsified APs using a single off-the-shelf Wi-Fi chipset without user interaction. We evaluate the effectiveness of the defense mechanism. We note that this mechanism also provides a reference to detect GPS spoofing attacks because it enables the device to obtain its current location with the identified legitimate APs.

Paper organization: The paper is organized as follows. We discuss the related work in Section II. In Section III, we briefly introduce the LLT inference attack and describe the restrictions on launching it. The privacy concerns regarding the LLT inference attack are discussed in Section IV. In Section V, we show how the revealed locations of APs in LLT enable an attacker to launch a successful spoofing attack in dense urban areas without jamming. Our findings are summarized into four criteria. We also design the attack methodologies to facilitate the attack in static and dynamic environments using these criteria, respectively. In Section VI, we discuss the countermeasures to mitigate the discovered spoofing attack. Evaluations are conducted in Section VII and we conclude the paper in Section IX.

II. Background

In this section, we introduce the preliminaries of Wi-Fi positioning systems, then the Google Geolocation API, and finally discuss the related work.

A. Wi-Fi Positioning Systems

The most widespread techniques of Wi-Fi positioning systems include range-based mode and range-free mode. Both modes use Wi-Fi APs as localization stations that broadcast service announcement beacons from known locations at a fixed interval (e.g., 102.4 milliseconds). In range-based mode, a mobile device detects localization signals transmitted by nearby Wi-Fi APs, records the Received Signal Strength (RSS) measurements, and sends the aggregated Wi-Fi information to the Wi-Fi positioning system, which then converts these RSS values into ranges and estimates the position of the client device based on these ranges. In range-free mode, the client device scans Wi-Fi APs in its reception range. The Wi-Fi positioning system then estimates the position of the client device based on the known locations of these visible Wi-Fi APs. Typically, both modes achieve positioning accuracy in the order of meters.

Location lookup table (LLT): Google Wi-Fi positioning system is a metropolitan area positioning system. It is a software-only system that requires a mobile device to have a WLAN-capable chipset and Internet connection. Google maintains the LLT, which contains estimated location data of Wi-Fi APs in the connected world. Google updates and extends its LLT by correlating GPS location data and the Wi-Fi information uploaded by a mobile device. For each Wi-Fi AP available in LLT, Google records its MAC address and an estimated geographic location.

Localization process: The localization process of Wi-Fi positioning systems can be divided into three steps as seen in Fig. 1. In step 1, a mobile device scans all 802.11a/b/g channels for visible Wi-Fi APs. It records their MAC addresses and signal strengths once detects these APs. In step 2, the mobile device queries the Google LLT server with the recorded MAC addresses

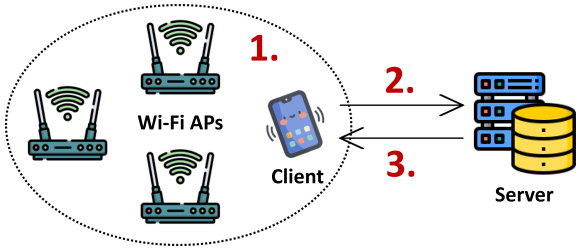


Fig. 1. Localization process of Wi-Fi positioning system: 1. The client device detects Wi-Fi APs in its reception range. 2. The client device queries the LLT server. 3. The server returns a position estimate based on the received Wi-Fi information.

and signal strengths over an encrypted channel. In step 3, the LLT server compares the reported MACs to the data residing in its LLT, computes a position estimate leveraging the estimated geographic locations of these MACs in LLT, and finally returns the position estimate to the mobile device. The construction of LLT depends on the mobile device of users. Whenever the mobile device runs navigation services (e.g., Google Maps, Uber, and Lyft), it actively collects nearby Wi-Fi information and sends all collected information to the location database along with GPS signals. The database then computes the location data of each Wi-Fi AP by correlating its information with the corresponding GPS signals. The database is updated regularly when Wi-Fi information is detected and uploaded.

B. Google Geolocation API

Geolocation API is one of the Places APIs hosted by Google Maps Platform. It returns a localization result based on the information about cell towers and Wi-Fi nodes that a client can detect. The communication is done over HTTPS using Power-On Self-Test (POST). Both request and response are formatted as JSON (an open standard file format). In this paper, each API request is composed of information about Wi-Fi APs. Due to privacy concerns of leaking location data of APs, the request payload `wifiAccessPoints` must contain at least two APs. For each AP, the field of `macAddress` is required, and all other fields are optional, including `signalStrength`, `age`, `channel`, and `signalToNoiseRatio`. A successful geolocation request returns a JSON-formated response containing the fields of `location` and `accuracy`, where `location` consists of an estimated geographic location with latitude and longitude, and `accuracy` indicates the accuracy of the estimated `location` in meters. In the case of an error, the API returns a JSON-formated error response. The error could be related to various reasons. For example, no location data in LLT for the received MAC addresses would result in an error indicating *no results were found even the request was valid*.

C. Related Work

Location Spoofing Attack: Spoofing attacks targeting mobile navigation systems were first discussed in [49]. The authors pointed out that malicious interference with civilian GPS signals can be a severe problem. Nowadays, it becomes increasingly feasible to build low-cost GPS spoofers that can generate fake GPS signals for any chosen location (e.g., [2], [3]). For example, recent studies show that the cost to create a portable and programmable spoofer can be less than $300 [35], [39]. Zeng et al. [59] first demonstrated the feasibility of stealthy fooling GPS-based road navigation systems. The attacker takes over the GPS signals of the victim device and manipulates the shape of a route shown on the navigation software to avoid being detected. One defense approach is to use inertial sensors to keep track of a vehicle's movement patterns. However, Narain et al. [38] revealed attacks against this approach. The attacker can fool the victim into driving on a route that yields inertial sensor readings similar to those from the original path.

Localization Services: According to our study, Google Maps determines the current location leveraging various location sources (e.g., GPS, Wi-Fi positioning systems, and GSM localization), and it shows the location of one source with the best localization accuracy. For example, it shows the position estimate from the Wi-Fi positioning system when the accuracy of GPS is worse than that of the Wi-Fi positioning system. The Wi-Fi positioning system is known as a complement to GPS when GPS signals are weak in urban canyon environments [36]. Therefore, to fool a victim in urban areas without raising alarms, it is equally important for an attacker to stealthily launch spoofing attacks on GPS and Wi-Fi positioning systems at the same time. Otherwise, the victim can be notified of the attacks as he/she is aware of the contradiction between localization results from GPS and Wi-Fi positioning systems. In addition to GPS and Wi-Fi positioning systems, the network location using GSM or IP address often returns the worse localization results with an accuracy of over 1,000 meters and can also be jammed or spoofed [40], [47]. In this work, we intend to address the inconsistency between GPS and Wi-Fi positioning systems when an attack is conducted in urban areas with dense coverage of Wi-Fi APs. We assume the locations from both GSM and IP address are either jammed or spoofed.

Countermeasures: In general, both cryptographic and non-cryptographic countermeasures were proposed to address GPS spoofing attacks (e.g., [22], [52], [5], [6], [11], [24], [25], [26], [42], [44], [51], [53], [57]). However, these techniques do not target the aforementioned advanced attacks (i.e., [38], [59]) and some of them need significant modifications to existing GPS infrastructure. Recently, Liu et al. [31] presented a new countermeasure to localize a GPS spoofer by iteratively moving towards the incoming angle of fake GPS signals. It derives the angle-of-arrival (AoA) of received signals by rotating a one-side-blocked GPS receiver. Other recent methods have focused on mitigating spoofing attacks against LiDAR systems of Autonomous Vehicle (AV) [7], [13], [15], [20], [20], [58]. These are out of the scope and will not be discussed further.

To mitigate signal spoofing attacks against Wi-Fi APs, several countermeasures have been proposed to detect unauthorized Wi-Fi APs [12], [28], [29]. Kohno et al. [27] first proposed the use of the clock skew of a computer on a network for the purpose of fingerprinting. The authors showed that the clock skew of a device remains consistent over time but it varies significantly across different devices, and thereby it can be used

as a reliable fingerprint to identify remote physical devices. On the wireless side, Jana et al. [23] proposed to determine the clock skew using Time Synchronization Function (TSF) timestamps in IEEE 802.11 beacon/probe response frames sent by Wi-Fi APs. It has been shown that it is possible to fool this fingerprinting mechanism by modifying the device driver of a Wi-Fi AP [8]. Alternatively, Hua et al. [21] proposed to fingerprint wireless devices using Carrier Frequency Offset (CFO) inferred from channel state information. The authors mention that CFO is an ideal fingerprint because it is attributed to the oscillator drift caused by the hardware imperfection and cannot be spoofed by any software [9], [10]. This mechanism needs to collect CFO from 5,000 frames in 10 seconds for constructing a high-precision fingerprint, and therefore may not be able to detect fake APs against Wi-Fi positioning since an AP usually only broadcasts 10 beacon frames in one second.

We also discuss the countermeasures to detect unauthorized Wi-Fi APs using physical-layer characteristics such as the angle-of-arrival (AoA) or time-of-flight (ToF) of the incoming Wi-Fi signals. However, existing research demonstrates that STO is the most degrading effect for estimating ToF and AoA. This is because the phase shift raised by STO in the frequency domain can cause circular rotation by the same amount in the time domain [46]. Xie et al. [55] eliminated STO by averaging approximately 200 CSI measurements collected from multiple consecutive channels within a short coherence time. To achieve this, [55] allocates over 200 MHz bandwidth, which is infeasible in common scenarios.

Fig. 2. Examples of the Google Geolocation API requests with two APs using the range-free mode and the range-based mode, respectively.

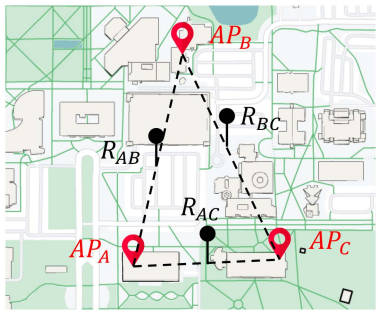


Fig. 3. LLT inference attack: The geographic locations of AP_A, AP_B, and AP_C in LLT calculated using the position estimates R_{AB}, R_{BC}, and R_{AC} from API requests with pairwise APs, i.e., (AP_A, AP_B), (AP_B, AP_C), and (AP_A, AP_C) using the range-free mode.

III. LLT Inference Attack

As mentioned earlier, we discover that the LLT inference attack can expose the geographic locations of APs in LLT from the Google location database. This enables privacy threats and allows an attacker to construct a location spoofing attack without relying on jamming in dense urban areas.

Brief overview of the LLT inference attack: Assume that the attacker is at location L_1 and he detects n APs from this location, namely AP_1, AP_2,..., AP_n. As discussed earlier, the attacker wants to know the precise locations that are recorded by LLT for these APs. Towards this goal, we find that the attacker can use an AP that the coarse location is far away from location L_1 (e.g., over 300 meters), and makes n API requests with this faraway AP and each of APs (this faraway AP must be available in LLT, but the attacker does not need to know its precise location). In particular, the attacker sends n API requests with (AP_1, faraway AP), (AP_2, faraway AP),..., and (AP_n, faraway AP) individually, and accordingly Google returns the attacker with n position estimates R_1, R_2,..., R_n. We discover that under certain conditions (see Section III-C), these position estimates can leak the LLT. In what follows, we discuss these conditions and the details of the attack. Note that, for each API request, the attacker only needs to fill `macAddress` and `signalStrength` for each AP. Other fields such as `channel` and `signalToNoiseRatio` are optional and they do not impact the localization results. The attacker can just rule out these fields.

A. Range-Free Mode

We start by demonstrating the LLT inference attack using the range-free mode as shown in Fig. 2(a). For the range-free mode, the payload `wifiAccessPoints` of a Geolocation API request only contains the field of `macAddress` of each AP. Recall that the range-free mode localizes a mobile device based on the known locations of APs in the reception range of the mobile device. The range-free mode does not use RSS values to localize a device, nor does it provide the estimated distances between APs and a mobile device in response to an API request.

We find that Google returns a position estimate that is the middle point on the line between the geographic locations of two APs in LLT by making an API request with these APs using the range-free mode. This means that even without the estimated distances, an attacker can compute the locations of APs in LLT using the position estimates obtained by making API requests with pairwise APs. In particular, as shown in Fig. 3, let AP_A, AP_B, and AP_C denote three APs detected from three different locations L_A, L_B, and L_C, respectively. By making API requests with pairwise APs, i.e., (AP_A, AP_B), (AP_B, AP_C), and (AP_A, AP_C) using the range-free mode, the Geolocation API returns the position estimates R_{AB}, R_{BC}, and R_{AC}, respectively, where R_{AB}, R_{BC}, and R_{AC} are the midpoints between the geographic locations of AP_A, AP_B, and AP_C, respectively. Then the attacker is able to compute the geographic locations of three APs

in LLT by solving the following equation:

$$\begin{bmatrix} X_A + X_B & Y_A + Y_B \\ X_B + X_C & Y_B + Y_C \\ X_A + X_C & Y_A + Y_C \end{bmatrix} = 2 \cdot \begin{bmatrix} X_{AB} & Y_{AB} \\ X_{BC} & Y_{BC} \\ X_{AC} & Y_{AC} \end{bmatrix} \quad (1)$$

where X_A, X_B, and X_C represent the latitude of AP_A, AP_B, and AP_C; Y_A, Y_B, and Y_C represent the longitude of AP_A, AP_B, and AP_C; X_{AB}, X_{BC}, and X_{AC} is the latitude of R_{AB}, R_{BC}, and R_{AC}; Y_{AB}, Y_{BC}, and Y_{AC} is the longitude of R_{AB}, R_{BC}, and R_{AC}, respectively. The six unknowns X_A, X_B, X_C, Y_A, Y_B, and Y_C can be solved from (1). They form the locations of these APs in LLT. The validation of these results is demonstrated in [18].

Update: We noticed that Google gradually changed the returned values of the Geolocation API in early October 2021. Consequently, after October 2021, instead of returning the middle point between two APs as the position estimate, the Geolocation API directly returns the geographic location of one out of two APs by making the same API request. We know this because we had already discovered the locations of multiple APs in LLT before October 2021. For example, we obtained the locations of AP_A and AP_B in LLT before October 2021. By making an API request with the same APs using the range-free mode after October 2021, the Geolocation API returned the position estimate R_{AB} that is only 0.86 meters away from the location of AP_A inferred before October 2021. We start with the LLT inference attack created before October 2021 because we can not validate the up-to-date LLT inference attack alone. In the meantime, the inferred locations of APs before October 2021 provide ground truth reference to the up-to-date LLT inference attack valid since October 2021.

B. Range-Based Mode

Although Google Geolocation API returns the location of one out of two APs in response to an API request with both APs using the range-free mode, the Geolocation API does not provide information regarding which AP is at the location of the position estimate. Nevertheless, by using the range-based mode, an attacker can discover this information. To be specific, the attacker can make an API request with pairwise APs using the range-based mode as seen in Fig. 2(b). For each AP in this API request, the attacker also includes a field of `signalStrength` which is the RSS value of the Wi-Fi signals sent from this AP. According to our tests, we find that *the Geolocation API returns a position estimate that is equal to the geographic location of the AP with a larger* `signalStrength`*value in the range-based mode*.

We form an API request using AP_A and AP_B. We fill in the fields of `signalStrength` for AP_A and AP_B with -45 and -55, respectively. By making this API request with these APs, the Geolocation API returns a position estimate that is the same as the geographic location of AP_A. Again, we make the same API request while switching the `signalStrength` values by using -55 and -45 for AP_A and AP_B, respectively. The returned position estimate is at the location of AP_B.

C. Restrictions on the LLT Inference Attack

We also look into the restrictions on conducting the LLT inference attack, especially using the range-based mode. We first describe the usage limits of the Google Geolocation API enforced by Google. We show that an attacker is capable of obtaining the geographic locations of a large number of APs in LLT each month at a negligible cost. Next, we demonstrate how the distance between the locations of two APs in an API request influences the effectiveness of the LLT inference attack. Finally, we present the conditions for the field of `signalStrength` of each AP in an API request to obtain a valid response in the range-based mode.

1) Usage of Google Geolocation API: Google Maps Platform products (e.g., the Geolocation API) are secured from unauthorized use by accepting API calls with valid authentication credentials. These credentials are in the form of an API key, which is a unique alphanumeric string associated with a Google billing account. To enable the Geolocation API, an attacker needs to apply a valid API key using one of its Google billing accounts and associate this API key with the Geolocation API.

The Geolocation API applies a pay-as-you-go pricing model. For each API key associated with a billing account, Google Maps Platform each month provides free credits of $200, which are automatically applied to qualified products such as the Geolocation API [32]. Google charges $5 per 1,000 API requests when one makes within 100,000 requests using the same API key each month, and the charge decreases to $4 per 1,000 API requests when an API key exceeds 100,000 API requests. Hence, a valid API key can make up to 40,000 free requests each month. This means that an attacker is capable of revealing the geographic locations of 40,000 APs in LLT for free by conducting the LLT inference attack using a single valid API key each month. To make more API requests for free each month, the attacker can apply as many valid API keys as it needs. Google Maps Platform does not restrict the maximum number of Geolocation API requests per day. Nevertheless, each API key is limited to 100 Geolocation API requests per second.

2) Restriction on the Distance Between Two APs: Google returns the location of the AP with a larger `signalStrength` in response to an API request with pairwise APs using the range-based mode. However, we observe in some rare cases that an API request returns a position estimate that is different from the location of the AP with a larger `signalStrength`. For example, although AP_A is assigned with a larger `signalStrength` value, by making an API request with AP_A and AP_C, the AP that we detect from location L_C in the neighborhood of location L_A, we obtain the position estimate R_{AC} that is about 54 meters away from AP_A. The distance between the locations of AP_A and AP_C is about 213.1 meters, and the distance between AP_A and AP_B is about 335.5 meters. This observation motivates us to investigate which conditions exactly enable the Geolocation API to return the location of the AP with a larger `signalStrength` value.

To address this question, we make use of 1,590 public APs collected around campus. For each AP, we make an API request with AP_A while assigning the `signalStrength` of AP_A and

this AP with -45 and -55, respectively. As a result, we observe the following scenarios:

1) The Geolocation API returns the position estimate equal to the location of AP_A when the `accuracy` is equal to 180 meters.
2) The Geolocation API returns a different position estimate close to the location of AP_A when the `accuracy` is less than 180 meters.

We then further examine when the Geolocation API returns the `accuracy` of 180 meters by making API requests with two APs using the range-based mode. In particular, we calculate the distances between the revealed locations of 1,590 APs and AP_A. We observe that *the Geolocation API returns an* `accuracy` *of 180 meters when the distance between two APs is over 300 meters.* And it returns an `accuracy` less than 180 meters when the distance is less than 300 meters. For example, an API request with AP_A and an AP randomly selected from the 1,590 APs obtains the `accuracy` of 97.1 meters and the distance between these APs is about 297.3 meters. By contrast, the distance between another randomly chosen AP and AP_A is about 312.3 meters, and the Geolocation API returns the `accuracy` of 180 meters.

All aforementioned observations seem to imply that there is no maximum distance between two APs to obtain the location of one out of two APs by making an API request with pairwise APs using the range-based mode. To further validate this implication, we make use of APs that are thousands of meters away from AP_A. These Wi-Fi APs are extracted from Wireless Geographic Logging Engine (WiGLE), a website that collects Wi-Fi APs that are used in the real world [54]. For each AP we obtain from WiGLE, we again make an API request with AP_A while assigning this AP with a larger `signalStrength` value. As a result, we successfully discover the geographic locations of these APs that we choose from WiGLE and are available in LLT, even though some APs are over 2,000 miles away from AP_A. This may indicate that the Google Geolocation API does not enforce a maximum distance between the locations of two APs for a valid response.

IV. Implication of the LLT Inference Attack on User Privacy

Our findings show how the LLT inference attack can reveal the locations of APs in LLT via API requests with two APs, especially regardless of the restrictions on the maximum distance between APs. In this section, we discuss the privacy issues that may be raised by the LLT inference attack.

A. Privacy Threat Example I: Monitoring Daily Activities of Residents

The feasibility of discovering the geographic location of an AP using the LLT inference attack also implies that the attacker knows the address of the apartment, house, building, office, etc that hosts this AP. We show that the attacker can achieve this by using the Google reverse-geocoding API, which is one of the core features of the Geocoding API provided by Google Maps Platform and converts a geographic location described by latitude and longitude into a human-readable address.

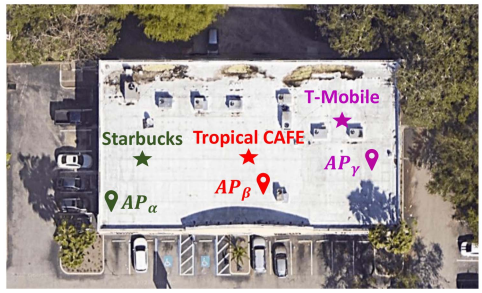


Fig. 4. Geographic locations of AP_α, AP_β, and AP_γ from Starbucks, Tropical Smoothie Cafe, and T-Mobile in the Google location database, respectively.

Experimental verification: We perform a trial of experiments to demonstrate the effectiveness of obtaining the correct street address given the revealed geographic location of an AP using Google reverse-geocoding API. Each reverse-geocoding API request must contain at least a field of `latlng`, consisting of the latitude and longitude values specifying the location on the map. A successful reverse-geocoding API request returns a JSON-formated response containing the field of `formated_address`, including a street address, postal code, and political entity (city, state, and country). In our experiments, we first discover the location of an AP in LLT using the LLT inference attack and then obtain the street address of the AP by feeding the inferred location into the reverse-geocoding API.

To avoid ethical issues, we take public Wi-Fi APs with recognizable Service Set Identifier (SSID), such as "McDonalds Free Wi-Fi", "Walmartwifi", and "Chick-fil-A Guest Wi-Fi". We detect 137 unique APs from 62 different locations in the city where we conduct this experiment. For each AP, we obtain the location in LLT using the LLT inference attack, as well as the corresponding street address by making a reverse-geocoding API request with the revealed location. As a result, we successfully identify the exact street addresses for 135 out of 137 APs. For a particular example, let AP_α, AP_β, and AP_γ denote 3 APs with SSIDs "Starbucks WiFi", "Tropical Smoothie", and "T-Mobile" from 3 stores in the same building of a mall, respectively. We can still differentiate these stores using the revealed locations of their respective APs in LLT as shown in Fig. 4. Among the 137 APs, the locations of 2 APs are not successfully converted by the reverse-geocoding API to valid addresses because both APs are located in areas under construction and Google did not update its Maps on time.

Potential privacy threat via the LLT inference attack: Existing research works have shown that the attacker can infer sensitive information (e.g., video content that they watch and demographics) through Wi-Fi traffic analysis [17], [30], [45], [50]. Given the capability of inferring the correct street address that hosts the AP, the LLT inference attack further deteriorates these attacks in that it enables the attacker to link the Wi-Fi traffic to the precise street address of a private household and discover the daily activities of its residents. For example, assume that the target AP is for a private household in the community with other APs and

households nearby. The attacker is unaware of the location of each AP and the associated household. The goal of the attacker is to identify the AP and the corresponding household, and then infer sensitive information about this household by sniffing the Wi-Fi traffic of this AP. Without the LLT inference attack, an attacker may have to localize each AP using specialized hardware (e.g., directional antennas) to measure the Angle-of-Arrival (AoA) or the Received Signal Strength (RSS) of Wi-Fi signals emitted from this AP on the scene. By contrast, the LLT inference attack directly infers the street address of this AP given its Wi-Fi MAC address with negligible costs.

B. Privacy Threat Example II: Monitoring Relocation

The attacker can keep track of the location of an AP no matter where it moves to by using the LLT inference attack. Consequently, the attacker can discover the relocation or travel of the user who carries and uses this AP.

For example, we first found such a movement in November 2021 while we were trying to discover the geographic locations of APs from the Meta headquarters in California using the LLT inference attack. These APs were initially made public in 2016 and are now publicly accessible in [19]. Other than obtaining the locations of APs at the Meta headquarters in California, we surprisingly obtained the location of an AP in Eppstein, Germany. Given that this AP was initially detected at the Meta headquarters in California 6 years ago in [19], we infer that it was moved from California to Germany by its owner. Google then updated its location in LLT to Germany.

Likewise, we also detected such movements for hundreds of APs. These APs were collected in January 2022 from a limited region near the university campus where we conduct this research. At that time, we obtained the locations of these APs in LLT. All APs were located close to the campus. However, in March 2022, we discovered that they were moved to different locations far away from where we detected them. For example, 237 APs were moved to different cities in the same state, and 36 APs were moved to different states across the U.S. We also noticed that one AP was moved to Malaysia. Because the Wi-Fi MAC address is a globally unique identifier assigned to each AP, we infer that these APs were potentially relocated with their owners during the last two months.

C. LLT Inference Attack Via Other Location Services

We also study the privacy promise of Wi-Fi positioning systems from other location service providers, such as WiGLE, Mozilla, and Skyhook Wireless. In other words, besides the Google location database, we investigate if these location services are vulnerable to the LLT inference attack.

1) Wigle: We note that WiGLE can return the coarse locations of APs in its database but the accuracy of these locations is low compared to that of the LLT inference attack discovered from the Google location database.

Similar to the Google Wi-Fi positioning system, WiGLE extends its Wi-Fi location database by allowing volunteers using the WiGLE app to upload GPS locations and Wi-Fi information, including SSIDs and MAC addresses. Note that WiGLE only has about 365,000 registered users actively collecting Wi-Fi APs [54].

WiGLE is the only Wi-Fi location database that allows a single MAC address lookup. By providing the MAC address of one AP, WiGLE returns the estimated location of the AP if it is available in its database. However, we found that many Wi-Fi APs available in the Google location database are not recorded by WiGLE. For example, among 274 relocated APs in Section IV-B, we only obtain the geographic locations of 140 APs from WiGLE. We also notice that WiGLE does not update its database as frequently as Google. Only 13 out of 140 APs available in WiGLE show similar relocation of the APs according to the obtained relocation of these APs from Google. Therefore, WiGLE is not suitable to discover the relocation of a user who carries and uses the same AP.

In addition, we find that the accuracy of the location from WiGLE for an AP is extremely low. For 137 public APs available in the Google location database mentioned in Section IV-A, we obtain the locations of 88 APs from WiGLE. However, we can only obtain the correct street addresses for 24 APs using their geographic locations from WiGLE. For a particular example, the geographic locations of AP_α, AP_β, and AP_γ from WiGLE are about 547.8 meters, 161.6 meters, and 533.5 meters away from the locations of these APs obtained from the Google location database shown in Fig. 4, respectively. Hence, we are not able to differentiate these stores by leveraging the locations of their respective APs from WiGLE. Besides, WiGLE API only allows a maximum of 20 queries daily. Therefore, an attacker can not utilize WiGLE to infer users' location privacy on a large scale like our LLT inference attack using the Google Geolocation API.

2) Mozilla Location Service: We identify that Mozilla location service is also vulnerable to the LLT inference attack.

The Mozilla Location Service (MLS) is an open geolocation service that localizes its customers based on visible Wi-Fi APs via its Geolocation API. It recorded over 2.3 billion Wi-Fi APs worldwide in July 2022 [37]. Identical to Google, each API request to Mozilla must contain at least two available APs for a valid response. Otherwise, an error is returned indicating no results were found.

We notice that the Mozilla location database returns a position estimate that is the middle point between the geographic locations of two APs in its database by making an API request with these APs using the range-free mode. Therefore, it is feasible for an attacker to infer the locations of APs in the Mozilla location database using the position estimates returned from API requests with pairwise APs.

3) Other Location Services: We attempt to investigate other location services, like Combain Positioning Service [4], Unwired Labs [34], and Skyhook Wireless [43]. However, none of these location service providers grant us access to their Geolocation APIs. Combain Positioning Service has discontinued its Wi-Fi positioning service for outdoor areas for new customers. Unwired Labs enable Wi-Fi positioning only for commercial implementations because of their privacy concerns regarding Wi-Fi location data. Skyhook Wireless was recently acquired by Qualcomm. Amidst the acquisition,

they are only serving enterprise partnerships at the time of writing.

Nevertheless, according to their API documentation, all the above location services offer Geolocation APIs similar to the Google and Mozilla Geolocation APIs. In other words, they all require at least two APs available in their databases for a valid response, and they all accept API requests using either the range-free mode or the range-based mode. Considering the common weakness in the Google and Mozilla location services, it is urgent for these location service providers to be aware of the LLT inference attack.

V. Manipulation of Google Wi-Fi Positioning System

As mentioned earlier, given the locations of APs in LLT, an attacker can intelligently conduct a location spoofing attack against the Google Wi-Fi positioning system without physical-layer jamming even though dozens of legitimate APs are present. Our findings are summarized into four criteria, which can enable an attacker to maximize its success rate in dense urban environments.

A mobile device usually provides all APs that it detects to Google to seek its location. Thus, unlike the LLT inference attack, a Geolocation API request for localization can include more than two APs. We assume that the API request consists of M_R legitimate APs and M_S falsified APs at the same time, where M_R and M_S contain at least two different APs available in LLT individually. We adopt the range-based mode for each Geolocation API request. Note that in this section, the range-based mode and the range-free mode yield similar results, and we adopt the range-based mode because it provides better location accuracy.

A. Initialization of the Attack

The manipulation of the Google Wi-Fi positioning system is motivated by the following question: Is it practical to launch a spoofing attack against the Google Wi-Fi positioning system in urban areas with dozens of legitimate Wi-Fi APs? We first study the constraints of existing attacks when a victim device carries out active scans or passive scans.

Active Scanning: A mobile device that uses an active scan broadcasts probe requests on a Wi-Fi channel and expects to receive probe requests from APs on this channel [1]. Consequently, an attacker is able to target this channel and replay the localization signals of falsified APs when the channel is idle or block the signals transmitted by legitimate APs when it senses activities on the channel.

Passive Scanning: In contrast to an active scan, a passive scan silently waits for signals broadcast by APs and a victim device does not actively send probe requests on any channel. This means that the attacker has to block all channels simultaneously. In the past, before IEEE802.11n was adopted, there were only 11 channels with approximately 80 MHz bandwidth and it may be possible for an attacker to jam all channels at the same time. However, with the wide adoption of IEEE802.11n, an AP can use over 30 channels with 1 GHz bandwidth nowadays, including channels on 2.4 GHz and 5 GHz bands [1]. Jamming all channels concurrently can cause a non-trivial cost for the attacker even if the attacker adopts reactive jammers [41], [48]. Moreover, jamming signals on Wi-Fi bands can significantly increase the chance for an attacker to be detected.

To sum up, passive scanning imposes a practical hurdle against traditional Wi-Fi spoofing attacks in real-world scenarios. Fortunately, although Android and Apple iOS do not provide a handler to switch between active scanning and passive scanning, both Android and Apple devices scan Wi-Fi passively if a user disables the Wi-Fi option, according to our observations. Such findings can force the attacker to perform location spoofing attacks in the context of passive scanning by jamming multiple possible Wi-Fi channels.

B. Criterion I

Assume Google receives Wi-Fi information about M_R legitimate APs from location L_R and M_S falsified APs from location L_S in the same API request. Satisfying $M_S \geq M_R + 1$ is sufficient enough for Google to return a position estimate at location L_S rather than location L_R.

To demonstrate Criterion I, we detect 48 and 25 APs from locations L_A and L_B, respectively. The distance between location L_A and L_B is over 300 meters. Let M_A and M_B denote the numbers of APs from location L_A and L_B in each API request, respectively. We examine when the Geolocation API returns a position estimate at location L_A by controlling M_A and M_B. More concretely, each API request contains 25 APs from location L_B, which means $M_B = 25$. In the meantime, we cumulatively add an AP from M_A to each API request. The field of `signalStrength` of each AP is assigned with a random RSS value. By making these API requests, we observe the following scenarios:

1) The Geolocation API returns a position estimate at location L_B when $M_B > M_A$. The position estimate remains the same when we make an API request with M_B alone.
2) The Geolocation API returns a position estimate at location L_A when $M_A > M_B$. The position estimate remains the same when we make an API request with M_A alone.
3) The Geolocation API returns a position estimate at a location between L_A and L_B when $M_A = M_B$.

The above results indicate that the number of APs plays a dominant role for Google in determining the position estimate when it receives mixed Wi-Fi information from different locations. We study if these results can be modified by changing the fields of `signalStrength`. Technically, we make API requests with the same set of APs but assign each AP with a designed `signalStrength` value. For example, suppose an API request consists of $M_A = 25$ and $M_B = 24$ APs. By intentionally assigning M_A APs with the `signalStrength` values of -80 while assigning the `signalStrength` values of M_B APs with -40, we wonder if such an API request will return a position estimate at location L_B rather than L_A. However, the position estimate is only slightly shifted away from location L_A (e.g., less than 50 meters). All the above scenarios still hold no matter how we modify the field of `signalStrength`.

We also perform an experiment to demonstrate Criterion I in real-world scenarios. We take a Google Nexus 6 smartphone

running Android 7 to an open area without surrounding Wi-Fi APs. The Nexus 6 uses cellular networks for internet connection. We turn off its GPS and thus Google Maps switch to the Wi-Fi positioning when Wi-Fi signals are detected. According to our tests, Google Maps shows the localization results with the best accuracy among GPS, Wi-Fi positioning system, and Cellular. Therefore, it shows the locations from the Wi-Fi positioning system when we turn off the GPS because the accuracy of Cellular is over 1,000 meters. Next, we impersonate 3 APs from location L_A and 4 APs from location L_B by using a low-cost Wi-Fi microchip ESP8266, respectively. As a result, Google Maps on Nexus 6 is spoofed to location L_B. The distance between location L_B and the open area is over 1,000 meters.

C. Criterion II

Assume Google receives mixed Wi-Fi information from different locations. In order to localize the client device, Google first clusters APs using a radius of 200 meters and then determines the position estimate based on a cluster with the maximum number of APs.

Although satisfying $M_S \geq M_R + 1$ is sufficient enough for an attacker to spoof a victim device to a location different from its current location without physical-layer jamming. It is still possible that an attacker does not have sufficient falsified APs from the spoofing location against legitimate APs. For instance, suppose the victim device is physically at location L_B surrounded by 48 real APs. The goal of an attacker is to spoof the victim device from location L_B to location L_A by impersonating 25 APs against 48 legitimate APs. Is it feasible to fool the device without jamming?

The most straightforward way to address this problem is to impersonate additional APs detected at different locations close to the spoofing location. For example, we notice that the Google Geolocation API returns the position estimate at location L_A rather than location L_B when it receives 48 APs from location L_B, 25 APs from location L_A, and 24 APs from location L_C at the same time, where L_C is close to L_A. Interestingly, the position estimate is at location L_A, even though the distance between location L_A and L_C is about 120 meters. By contrast, the distance between location L_B and L_C is about 330 meters. This observation raises another question, i.e., how does Google determine the position estimate when it receives Wi-Fi information from multiple locations? Is it practical for the attacker to understand the localization routines adopted by Google and then intelligently impersonate APs from the spoofing location against legitimate APs?

We address this question using Criterion I. Specifically, given M_R legitimate APs from location L_R and M_S falsified APs from location L_S, where $M_R = M_S$, the Geolocation API returns a position estimate between the two locations by making an API request with M_R and M_S using the range-based mode. Our goal is to find out, under which circumstance, the Geolocation API returns the position estimate at location L_S rather than a location between L_R and L_S by adding an additional falsified AP close to location L_S.

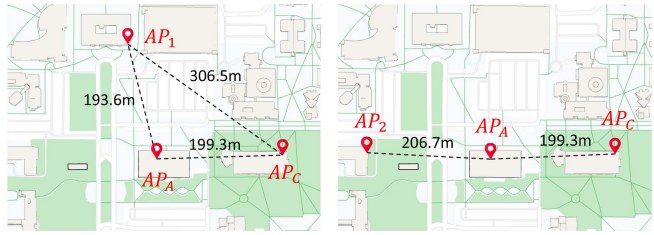


Fig. 5. Geolocation API requests with 5 APs, including AP_i close to L_A, $M_A = 2$ APs (i.e. AP_A and AP_C) from L_A, and $M_Z = 2$ APs from L_Z: (a) returns a position estimate at location L_A by taking AP_1 as AP_i and (b) returns a position estimate between location L_A and L_Z by taking AP_2 as AP_i.

We start by demonstrating the radius of 200 meters in Criterion II. We take $M_A = 2$ APs around location L_A as falsified APs. M_A consists of AP_A and AP_C shown in Fig. 5. Meanwhile, We take $M_Z = 2$ APs from location L_Z as legitimate APs. The distance between location L_A and L_Z is over 2,000 miles, while the distance between two legitimate APs is about 8.6 meters. AP_i, an additional falsified AP close to location L_A, is taken from the 1,590 public APs detected while driving 1.6 miles along a route starting from location L_A, i.e. $i = 1, 2, \ldots, 1,590$. For each additional falsified AP, we make an API request with 5 APs, including AP_i, APs in M_A (i.e., AP_A and AP_C), and APs in M_Z. The `signalStrength` values of the 5 APs are assigned with the random RSS values. As a result, we observe the following scenarios:

1) The Geolocation API returns the position estimate at location L_A if at least two mutual distances among the locations of AP_A, AP_C, and AP_i are less than 200 meters.
2) The Geolocation API returns the position estimate between location L_A and L_Z if two mutual distances among the locations of AP_A, AP_C, and AP_i are greater than 200 meters.

For example, in Fig. 5(a), if we take AP_1 as an additional falsified AP in an API request, the Geolocation API returns the position estimate at location L_A. The distances between 2 APs of two pairs i.e., (AP_1, AP_A) and (AP_1, AP_C) are about 193.6 meters and 199.3 meters, respectively. By contrast, in Fig. 5(b), the Geolocation API returns a position estimate between location L_A and L_Z by taking AP_2 as an additional falsified AP. In this case, only the distance between AP_A and AP_C is less than 200 meters. Obviously, 3 APs in Fig. 5(a) form a cluster of falsified APs $\hat{M}_A$ around location L_A as both AP_1 and AP_C are within a radius of 200 meters from AP_A. As a result, Geolocation API returns the position estimate at location L_A because $\hat{M}_A > M_Z$ in the same API request.

To further demonstrate it, we perform a trial of experiments using dozens of APs, shown in Fig. 6. Our strategy still uses Criterion I. We take $M_A = 60$ falsified APs from location L_A and $M_Z = 60$ legitimate APs from location L_Z, respectively. All APs in M_Z fall into a circle that has a radius of 200 meters centers at the location of one AP. In the meantime, Fig. 6(a) shows the distribution of APs in M_A from location L_A. We can see that 58 APs fall in the circle that centers at AP_3 with a radius of 200 meters.

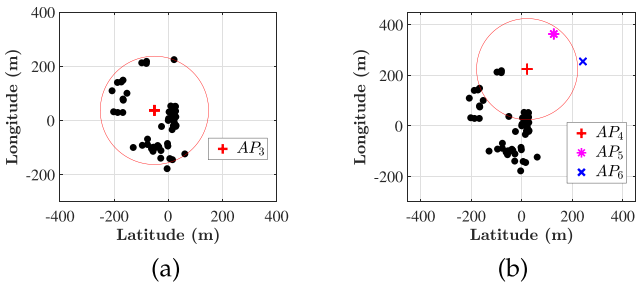


Fig. 6. Geolocation API requests with 121 APs, including AP_i close to L_A, $M_A = 60$ APs from L_A, and $M_Z = 60$ APs from L_Z: (a) the distribution of M_A APs from L_A and (b) API requests by taking AP_5 and AP_6 as an additional falsified AP AP_i individually.

First, we make sure that the Geolocation API returns a position estimate between the location L_A and L_Z by making an API request with M_A and M_Z. We then take an additional falsified AP AP_i close to location L_A from the 1,590 APs. For each additional falsified AP AP_i, we again make an API request with M_A falsified APs and M_Z legitimate APs using the range-based mode. As a result, we notice that given APs from different locations, Google likely adopts a clustering algorithm (e.g., DBSCAN [14]) to cluster APs using a radius of 200 meters and then determines the position estimate based on a cluster with the maximum number of APs. For instance, in Fig. 6(b), an API request returns the position estimate at location L_A by taking AP_5 as an additional falsified AP AP_i. The distance between AP_4 and AP_5 is about 188.2 meters, and AP_4 is one of the 60 falsified APs in M_A. Nevertheless, an API request taking AP_6 as an additional falsified AP AP_i returns the position estimate between location L_A and L_Z because the distance between AP_4 and AP_6 is over 200 meters. The red circle is the circle centers at AP_4 with a radius of 200 meters. Only 12 out of 60 APs in M_A fall in this circle.

However, we can not explicitly identify the black-box localization algorithms adopted by the Google Wi-Fi positioning system using Geolocation API. Ultimately, in Section VII, we show that the radius of 200 meters is sufficient for an attacker to maximize its success rate of location spoofing attacks in dense urban areas without jamming.

D. Criterion III

Given a number of APs with identical geographic locations in LLT, Google treats these APs as a single AP in determining the position estimate when Google receives them in one Geolocation API request.

We find that Google enforces this restriction starting from October 2021. Recall that each Geolocation API request must contain at least two APs available in LLT for a valid response. The API returns an error if the request consists of multiple APs located at one unique geographic location since October 2021. Note that the API used to treat such a request as valid and returned the location before October 2021. It seems that Google is trying to mitigate the leakage of Wi-Fi location data by enforcing this restriction, which, however, makes it easier for an attacker to launch the location spoofing attack. It is possible that the attacker can successfully deceive the localization of a victim device even with a fewer number of falsified APs than that of legitimate APs around the victim device. For example, assume 24 legitimate APs are located at 3 unique geographic locations near location L_A. To spoof the victim device from location L_A to L_B, the attacker only needs to impersonate 4 APs at different geographic locations near location L_B. In other words, the attacker can successfully spoof a victim device by using 4 falsified APs against 24 legitimate APs.

Google assigns the same location to Wi-Fi APs hosted by the same hardware device with slightly different Wi-Fi MAC addresses. For example, APs with MAC addresses of `"00:a2:ee:a3:xx:2c"`, `"00:a2:ee:a3:xx:5f"`, and `"00:a2:ee:a3:xx:17"` are associated with the same geographic locations in LLT. All MAC addresses are intentionally modified for anonymity and ethical concerns.

E. Criterion IV

Google discards APs not timely recorded in LLT in determining the position estimate due to the missing locations.

We find that the Geolocation API provides helpful information to an attacker on whether an AP is recorded by Google LLT. The attacker can simply make an API request with an AP that is known to be available in LLT while assigning an unknown AP with a greater `signalStrength` value. The Geolocation API returns the geographic location of this unknown AP if it is recorded in LLT. Otherwise, the Geolocation API returns an error "valid request format but no results were found". This information is important for the attacker because it enables the attacker to filter APs not recorded by Google LLT and construct a much more effective AP set to launch the location spoofing attack.

F. Attack Methodology

Given the LLT inference attack and 4 Criteria discovered from the Google Wi-Fi positioning system, it is straightforward for an attacker to construct the location spoofing attack against the Google Wi-Fi positioning system in dense urban areas without physical-layer jamming. In this section, we demonstrate our attack methodologies in static mode and dynamic mode, respectively.

1) Static Mode: In static mode, we assume that the victim stays at location L_R, and the list of legitimate APs visible to the victim device does not change. The goal of the attacker is to spoof the victim device from location L_R to the spoofing location L_S. The spoofing attack consists of 3 stages as discussed below:

Stage 1: The attacker detects legitimate APs around location L_R and then discovers the geographic locations of these APs using the LLT inference attack. The attacker also identifies effective legitimate APs, which are recorded in Google LLT and have unique geographic locations. Let M_R indicate the number of these APs.

Stage 2: Similarly, the attacker uses the LLT inference attack to discover the falsified APs around location L_S and identifies the effective falsified APs by discarding APs that are not in LLT and treating APs with duplicated locations as a single AP. From

the set of effective falsified APs, the attacker further identifies APs that fall in the circle that centers at location L_S and has a radius of 200 meters. Let M_S indicate the number of these APs. We use a radius of 200 meters as a conservative way to intelligently select falsified APs from location L_S according to Criterion II.

Stage 3: The attacker crafts Wi-Fi signals of M_S falsified APs from location L_S. It then broadcasts crafted signals to mimic M_S falsified APs at location L_R.

In the case that $M_R > M_S$, we argue that jamming attacks are still necessary to conduct a successful attack. Nevertheless, in Section VII-B, we state that $M_S > M_R$ is common in real-world scenarios according to our experimental results. In addition, we discuss the pros and cons of the traditional location spoofing attack and the one discovered in this paper in Section VII-B.

2) Dynamic Mode: We further extend the attack methodology to accommodate dynamic scenarios. In dynamic mode, we assume the victim is moving in dense urban areas. In this case, the list of legitimate APs visible to the victim device is updated frequently. To successfully spoof the victim device, the attacker must update its list of falsified APs accordingly when the victim device conducts a fresh Wi-Fi scan at different locations. We then move to investigate the scanning frequency when the navigation software is running as a foreground app.

Frequency of Wi-Fi Scans: Historically, Google allowed an Android device to scan nearby Wi-Fi APs one time in 5 seconds. To improve the battery efficiency, this scanning frequency is restricted to four times in 2 minutes for each foreground app since Android 9 [33]. Android also provides a developer option named *Wi-Fi scan throttling* to toggle the throttling off for testing purposes.

We use Google Nexus 6P and Pixel XL smartphones to conduct the measurements. The navigation software (e.g., Google Maps, Uber, and Waze) is running as a foreground app and the *Wi-Fi scan throttling* is on. We measure the frequency of active scans because active scans periodically broadcast probe request frames on each WLAN channel. As a result, we observe that both devices scan Wi-Fi one time in 5 seconds irrespective of the throttling. In contrast, a prototype Wi-Fi scanner built by us scans Wi-Fi one time in 30 seconds if the throttling is on. In addition to Android devices, we notice that Apple also scans Wi-Fi one time in 5 seconds when Apple Maps is running as a foreground app.

In summary, the attacker is recommended to detect the legitimate APs visible to the victim device and update its list of falsified APs accordingly every 5 seconds in dynamic mode. Other than this, the attacker follows the attack methodology in static mode.

VI. Countermeasures

One key application of the location spoofing attacks is to mislead a car navigation system, which usually relies on GPS and Wi-Fi signals for localization results. Past research has shown that it is possible for an attacker to spoof the GPS localization of a navigation system (e.g., [59]). To minimize the discrepancies between the localization results from GPS and Wi-Fi positioning systems in urban areas when the GPS spoofing attack is conducted, an attacker may also want to subvert the Wi-Fi localization by launching the location spoofing attack discovered in this paper.

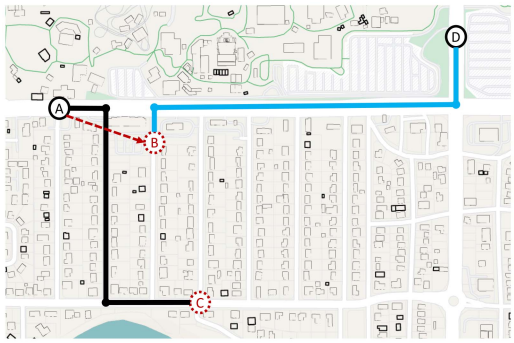


Fig. 7. Attack example: the victim is at location A and its destination is at location D; The spoofer sets the localization result of the victim device from A to B and forces the navigation system to generate a new route B → D (blue line). By following the navigation trajectory, the victim arrives at location C along route A → C (black line) in the physical world.

For traditional location spoofing attacks against Wi-Fi positioning systems, one effective defense would be detecting suspicious jamming signals. The location spoofing attacks discovered in this paper, however, cannot be addressed in this way because the attacker does not need to jam Wi-Fi signals from legitimate APs. In what follows, we propose potential countermeasures to mitigate this attack.

A. Threat Model

We start with the threat model raised by the location spoofing attacks in the context of on-road navigation.

We consider that the victim requests a ride from ride-sharing platforms (e.g., Uber, Lyft) in an unfamiliar area with a high density of Wi-Fi APs. The victim visiting the unfamiliar area heavily relies on mobile navigation systems for current localization results. We assume that the attacker is powerful enough to generate and transmit any spoofing signals (e.g., GPS and Wi-Fi positioning system) corresponding to any locations or navigation routes of his choice. The attacker can be a malicious Uber/Lyft driver who places the spoofers in the same car. The goal of the attacker is to stealthily manipulate the location and navigation trajectory shown on the victim device and divert the victim to a target destination pre-defined by itself for robbery. The attacker can adopt the spoofing path generators proposed in [38] and [59]. More specifically, given an original path from the starting point to the destination computed by the navigation software, the attacker computes a spoofing path from the current location to its desired destination, which has a similar shape as the original path. An attack example is shown in Fig. 7. To minimize the discrepancies between GPS and Wi-Fi positioning systems along the spoofing path, the attacker always spoofs the GPS and Wi-Fi positioning systems concurrently to the same spoofing location. Our goal is to detect this location spoofing attack.

B. Using Reference APs

The most straightforward way to detect the existence of an attack is to infer the geographic locations of all visible APs using the LLT inference attack. Nevertheless, such a method raises privacy concerns and needs to consume a tremendous amount of API requests. It is also possible to cause denial-of-service to the Google location database, particularly acknowledging that mobile devices scan Wi-Fi one time in 5 seconds. Alternatively, our strategy is to detect the existence of spoofing attacks leveraging Criterion I.

Motivation: Our intuition is that the victim device always detects mixed Wi-Fi information from two distinct locations when an attacker is present according to the attack methodology. To detect the presence of an attack, instead of making an API request with visible APs solely, a mobile device can make an API request with nearby APs that are detected at its current location, and a set of reference APs, which are pre-collected from a reference location far away from the current location. Let M_C and M_V indicate the numbers of the detected APs and the reference APs in an API request, respectively, where we set $M_C = M_V + 1$. And $M_C = M_R + M_S$, where M_R and M_S are the numbers of legitimate APs and falsified APs from the current location L_R and the spoofing location L_S, respectively. The position estimate of M_V is at location L_V. Note that M_V and L_V of the reference APs should be kept confidential.

Strategy: Recall that when Google receives mixed Wi-Fi information from different locations in the same API request, Google returns a position estimate that is the location surrounded with the largest number of APs. By making an API request with M_C and M_V, Google returns the position estimate at the current location L_R if there is no attack. This is because $M_C = M_R$ as $M_S = 0$, then $M_R > M_V$ in the API request as $M_R = M_V + 1$. On the other hand, Google returns the position estimate at location L_V if there is an attack and $M_S \in [2, M_C - 2]$. To explain it, first, $M_S \in [2, M_C - 2]$ means $M_R \in [2, M_C - 2]$ because $M_C = M_R + M_S$. Given $M_C = M_V + 1$, we can derive

$$\begin{cases} M_S = M_V + 1 - M_R \leq M_V, \ M_R \geq 2, \\ M_R = M_V + 1 - M_S \leq M_V, \ M_S \geq 2. \end{cases} \tag{2}$$

Therefore, Google returns the position estimate at location L_V as M_V has the largest number of APs in the same API request. To further demonstrate this method, we consider an example in which the mobile device detects $M_C = 21$ APs at location L_R, and we set $M_V = 20$ APs accordingly. If there is no attack, $M_S = 0$ and $M_R = 21$ APs. Because $M_R > M_V$, an API request with M_C and M_V returns the current location L_R. On the contrary, assume that an attack is present and the attacker broadcasts $M_S = 15$ falsified APs. Although the attacker successfully spoofs the mobile device to the spoofing location L_S as $M_R = 6$ and $M_S > M_R$, an API request with M_C and M_V returns the position estimate at location L_V because $M_V > M_S > M_R$. Thus, an alarm that indicates the existence of the location spoofing attack can be raised to the device because the above position estimate indicates that M_C consists of mixed Wi-Fi information from different locations.

Vulnerability: This countermeasure is lightweight as it detects the presence of spoofing attacks using fewer API requests. But it is vulnerable to *Denial-of-Service* attack.

We use an example to demonstrate the weakness. Assume that a victim device adopts it to detect the existence of spoofing attacks. Meanwhile, an attacker seeks to force the victim to disable this countermeasure and then spoof the device. To achieve it, the attacker can fake a small number of falsified APs (e.g., $M_S = 2$) consistently. Although the navigation software shows the correct location of the victim as $M_R > M_S$, it still triggers alarms because $M_V > M_R$. This may eventually force the victim to disable the countermeasure as it generates too many false alarms. In addition, we observe that there are some APs nonexistent in LLT. According to Criterion IV, these APs also raise alarms as they are discarded during the decision process.

To address this vulnerability, we explore differentiating legitimate APs from falsified APs using physical-layer features of Wi-Fi signals.

C. Using Physical Layer Features

In this subsection, we demonstrate another countermeasure to differentiate legitimate APs from falsified APs leveraging the variations of Symbol timing offset (STO) experienced by Wi-Fi signals in the context of car navigation. We show that this countermeasure enables the device to identify legitimate APs using a single off-the-shelf Wi-Fi chipset without user interaction.

Motivation: The intuition of this countermeasure is the characteristics of STO. According to the extensive measurements, we observe that *STO changes dramatically when the wireless channels undergo variations and vary slowly when the channels become stable*. Thence, it is possible to estimate the attenuation loss and distortion by measuring the variations of STO experienced by Wi-Fi signals. We assume the countermeasure is adopted in the context of car navigation. The Wi-Fi spoofer is placed within the same car as the victim device. In the meantime, legitimate APs transmit localization signals at different locations outside the vehicle. Hence, because the Wi-Fi spoofer is closer to the victim device, the detected Wi-Fi signals emitted from legitimate APs may undergo significant distortion. In other words, the STO experienced by these Wi-Fi signals may change more dramatically than the Wi-Fi signals transmitted by the Wi-Fi spoofer. Thus, we study the feasibility to differentiate legitimate APs and falsified APs leveraging the variations of STO measured from detected Wi-Fi signals.

Overview of STO: As a mature and spectrum-efficient multiplexing access technology, orthogonal frequency-division multiplexing (OFDM) has been widely adopted in Wi-Fi standards. However, OFDM suffers from frequency and timing synchronization problems due to hardware imperfection [?]. STO is the most degrading effect compromising the received signals. It emerges as it is not a priori knowledge when to expect a packet [46]. Because OFDM systems function on blocks of (time-domain) samples, named symbols, it is crucial that the right block is fed to the demodulator for correct decoding at the receiver side [?]. Each packet consists of multiple OFDM

symbols and a header with a known, periodic sequence named short-training field (STF). To find out the packet boundary as well as the boundary of each OFDM symbol, an auto-correlator/cross-correlator is utilized to detect the presence of Wi-Fi signals leveraging STF. However, due to the limited length of STF, receivers can not precisely eliminate errors in determining symbol boundary, leading to irreversible errors such as inter-carrier interference (ICI) and inter-symbol interference (ISI) due to the symbol timing offset τ_d. According to the existing research, τ_d varies in each packet reception but follows a Gaussian distribution with the zero means [?], [55]. In this work, we measure the variations of τ_d using channel state information (CSI).

Decomposing CSI phases: Channel State Information represents the combined effect of, e.g., shadowing, fading, and scattering during signal propagation through the wireless channels. By sending a packet with a known training sequence, the receiver captures the signal strength and phase information of each OFDM subcarrier to obtain an estimate of the channel. We focus on phase information of CSI measurements because STO causes phase shifts in CSI measurements. We denote N as the number of subcarriers in one Wi-Fi band. Assume H as the channel matrix and a complex vector represents the CSI measurements at different subcarriers. The CSI measurement for the kth subcarrier can be expressed as $H_k = |H|e^{-j\phi_k}$, where ϕ_k is the phase offset on subcarrier k. According to [?], [55], the reported CSI phase value ϕ_k from sub-carrier k can be expressed as

$$\phi_k = \psi_k + k \cdot (\lambda_d + \lambda_s) + \varphi \tag{3}$$

where λ_d is the phase shifts introduced by the time offset τ_d due to STO. It can be shown to impact CSI at k^{th} subcarrier as $H_k = |H|e^{-j \cdot k \cdot \lambda_d}$, where $\lambda_d = 2\pi \cdot \tau_d / N$. The other three components are caused by the following factors:

ψ_k: the phase rotation caused by the *time of flight* (ToF). This phase rotation is caused by channel propagation delay from the transmitter (Tx) to the receiver (Rx). In the absence of multipath for simplicity, the phase shift of subcarrier k caused by ToF can be expressed as $\psi_k = 2\pi f_k \tau_l$, where τ_l is the line-of-sight (LoS) propagation time and f_k is the center frequency of the k^{th} subcarrier.

λ_s: the phase errors caused by the *sampling frequency offset* (SFO). SFO is caused by the offset of the sampling frequencies between the Tx and Rx. Such a frequency offset also introduces a time shift τ_s which raises phase errors that linearly increase with the subcarrier index k. Therefore, λ_s can be expressed as $\lambda_s = 2\pi\alpha\tau_s$, where α is a constant. As the sampling frequency difference stays stable in the order of minutes [23], τ_s is considered as a constant value across different CSI sequences transceived between a pair of the transmitter and receiver.

φ: the phase error caused by the *carrier frequency offset* (CFO). Ideally, the carrier frequency f_c should be the same between each Tx-Rx pair for OFDM systems. However, there exists an offset between the Tx-Rx oscillators because of the hardware imperfection. Although NIC hardware compensates for this offset to reduce the noise in advance, there still exists a fractional CFO, $\triangle f_c$. This fractional CFO results in a phase error φ which can be represented as $\varphi = 2\pi \triangle f_c t$. Note that CFO is independent of subcarrier index k. According to the above analysis, (3) can be written as

$$\phi_k = 2\pi f_k \tau_l + k(\lambda_d + \lambda_s) + 2\pi \triangle f_c t \tag{4}$$

Estimating STO from CSI: Our goal is to measure the variations of λ_d across CSI measurements received from the same Wi-Fi AP. We adopt the same method mentioned in [55] to calculate λ_d. Specifically, we calculate the mutual phase differences of CSI measurements and obtain a set of $\triangle\psi_k + k \cdot (\triangle\lambda_d + \triangle\lambda_s) + \triangle\varphi = k \cdot \triangle\lambda_d + a$ for each subcarrier k, where a is a constant. We can obtain a set of $k \cdot \triangle\lambda_d + a$ because λ_s is a constant and can be removed by the deduction. In the meantime, we also assume that the wireless channels are static and the phase rotation ψ_k of ToF is a constant. Note that the wireless channels between legitimate APs and the victim device are not static. The mutual difference between ToF is combined into the mutual difference of λ_d. Next, we turn to measure the variations of $\triangle\lambda_d$ instead of λ_d. This is because if $\lambda_d \sim N(0, \sigma^2)$, then $\triangle\lambda_d$ should be a Gaussian with the zero mean as well.

We make use of a Raspberry Pi B4 to measure the variations of $\triangle\lambda_d$ of wireless signals. Particularly, for each visible AP, we collect its localization signals (i.e., beacon frames) using Raspberry Pi. To obtain the CSI measurement of each beacon frame, we enable the CSI extraction on Raspberry Pi using modified firmware in[?]. To demonstrate the motivation, e.i. STO changes dramatically when the wireless channels undergo variation and it changes slowly when the channels are static, we perform two trials of experiments as shown in Fig. 8. We take one Wi-Fi AP to an open area, and this AP broadcasts one beacon frame every 100 milliseconds on Channel 11. We then collect these frames using Raspberry Pi when the distance between Raspberry Pi and Wi-Fi AP is about 1 m and 10 meters, respectively. For each trial, we collect 600 CSI measurements and calculate the mutual difference between the phase values of CSIs. First, we show the distribution of $\triangle\lambda_d$ in Fig. 8(a). Both lines show Gaussian distributions with zero means. The x-axis in Figure is divided into 100 bins within the range of [-0.3, 0.3] and it shows the frequency of $\triangle\lambda_d$ falling into each bin on the y-axis. The figure shows that the variation of $\triangle\lambda_d$ when the distance is 1 m is considerably greater than that when the distance is 10 meters. In addition, we show $k \cdot \triangle\lambda_d$ versus the sub-carrier index k for 8 randomly selected CSI measurements when the distance is 1 m and 10 meters in Fig. 8(b) and (c), respectively. As a result, the $\triangle\lambda_d$ when $d = 1$ meter is greater and it changes dramatically than that when $d = 10$ meters. Thus, we deem that one can differentiate legitimate APs from falsified APs leveraging the variations of $\triangle\lambda_d$ of Wi-Fi signals.

Countermeasure Methodology: Given the characteristic of $\triangle\lambda_d$, we design the countermeasure methodology to differentiate legitimate APs. Assume the victim device is at location L_R and it obtains location L_C from the navigation software. The goal is to verify if L_C matches L_R. The methodology consists of 3 stages as discussed below:

Stage 1: The device first detects if it receives mixed Wi-Fi information from different locations. This can be achieved using

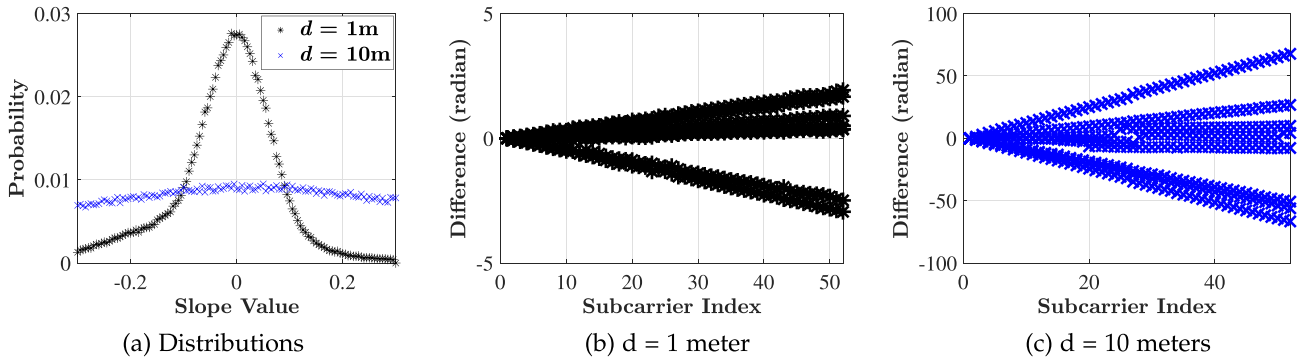


Fig. 8. Phase differences across sub-carriers and the distribution of $\triangle\lambda_d$. (a) Distributions of slope $\triangle\lambda_d$ when $d = 1$ and $d = 10$ meters, respectively; (b) The phase difference $k \cdot \triangle\lambda_d$ of 8 CSIs when $d = 1$ meter; (c) The phase difference $k \cdot \triangle\lambda_d$ of 8 CSIs when $d = 10$ meters.

reference APs. If it returns the position estimate of the reference APs, the device proceeds to the next stage.

Stage 2: The device requests the Wi-Fi chipset to sniff Wi-Fi signals on a specific channel and extracts CSI measurements. It calculates the mutual difference among CSI measurements from the same APs and measures the distributions of $\triangle\lambda_d$. Assume we take ϵ as the threshold to differentiate legitimate APs. Thus, one can take an AP as a legitimate AP if its variation of $\triangle\lambda_d$ is greater than ϵ.

Stage 3: When the device identifies at least two legitimate APs, it makes a Geolocation API request with these APs to the Google location database and obtains the correct location L_R. It then matches L_R with L_C shown on the navigation software. If these locations deviate, it raises an alarm indicating that the current location is spoofed.

The attacker may decide to fake the characteristics of STO by modifying the short-training field (STF) of the Wi-Fi signals it transmits. However, it requires the attacker to deploy specialized hardware like USRP. In addition, there is no existing research stating if the receiver is still able to recover the Wi-Fi signals even if the STF is modified.

VII. Spoofing Attack Evaluation

We evaluate the effectiveness of our location spoofing attack discovered in this paper in dense urban areas without jamming. We collect public APs by conducting Wi-Fi scans every 5 seconds while driving 20 different routes in the city where we conduct this research. These routes contain 10 different routes around a university campus and 10 different routes in the downtown area. The distance between the downtown and the campus is about 9 miles.

A. Evaluation of the Traditional Attack

We start by evaluating the traditional Wi-Fi location spoofing attack demonstrated in [19], [47]. Our strategy is that, given legitimate and falsified APs from location L_R and L_S, respectively, we evaluate if the attacker is able to fool a victim device from location L_R to L_S by making an API request to Google with both legitimate and falsified APs using the range-based mode.

We take APs detected from 10 different routes close to the university campus as legitimate APs. For each route, we conduct n Wi-Fi scans. Let M_R^i denote the set of legitimate APs visible during the i-th scan along this route, where $i = 1, 2, \ldots, n$. In the meantime, we take M_S^i as the set of falsified APs visible during the i-th scan along a route in the downtown area. Let L_R^i and L_S^i indicate the locations where we detect the APs in M_R^i and M_S^i, respectively. To maximize the success rate of this attack, we make the size of M_S^i twice that of M_R^i. For each trial of the experiments, we then make n API requests with the corresponding APs in M_S^i and M_S^i for $1 \leq i \leq n$.

As a result, we make 348 API requests in total and only 187 out of 348 API requests return the position estimates in the downtown area. We obtain the precise locations of these APs using the LLT inference attack, and we find that many APs from the downtown area have the same geographic locations in LLT. For a particular example, an API request with $\|M_R^i\| = 24$ and $\|M_S^i\| = 48$ returns the position estimate around the university campus. This means that the attacker fails to deceive the victim device around campus to get a spoofed position estimate chosen by the attacker in the downtown area. By looking into the locations of these APs in LLT, we find that the 24 legitimate APs are composed of 18 APs with different geographic locations, 3 APs associated with one unique geographic location, and 3 APs that are not recorded in LLT. Hence, 24 legitimate APs form a set with 19 effective APs. By contrast, the 48 falsified APs consist of 11 APs with different geographic locations, 24 APs located at 7 unique geographic locations, and 13 APs are not recorded in LLT. Namely, 48 falsified APs form a set with 18 effective APs. Consequently, the Geolocation API returns the position estimate close to the university campus.

B. Evaluation of the Discovered Attack

To evaluate the effectiveness of the discovered attack, we first obtain the geographic locations of APs using the LLT inference attack. We then discard APs that do not exist in the LLT and select APs with unique locations in LLT. As a result, 6,871 out of 19,993 APs are chosen as eligible APs for the evaluation. We further choose falsified APs according to the attack methodology discussed in Section V-F.

Again, our goal is to fool the victim device from its current location around campus to a location near the downtown area. We take APs detected from different routes close to the campus

and in the downtown area as legitimate and falsified APs, respectively. For each trial of our experiments, we make n API requests with the corresponding APs in M_R^i and M_S^i. Considering the overwhelming number of APs in the downtown area, we limit the number of falsified APs by setting $\|M_S^i\| - \|M_R^i\| < 10$ in each API request. As a result, all the 348 API requests return the position estimates in the downtown area. We also switch the legitimate and falsified locations, i.e., we use APs detected from routes close to the downtown area and the university campus as legitimate and falsified APs, respectively. We find that 3 trials of attacks fail because of the insufficient number of falsified APs. For example, in one out of three failed trials, location L_S is around an industrial park with $M_S = 52$ falsified APs, whereas the corresponding location L_R in the downtown area has $M_R = 64$ legitimate APs.

The above results show that the attacker has sufficient falsified APs against legitimate APs in most trials (e.g., 693 out of 696 trials). In rare cases where $M_R > M_S$, we argue the traditional attack needs to block as many legitimate APs as it can to maximize its success rate. According to our collections, 70% APs are transmitting on 6 different channels on both 2.4 GHz and 5 GHz bands in most trials. Consequentially, the traditional attack has to jam at least 6 channels by using multiple jammers simultaneously. Compared to the traditional attack, the one discovered in this paper allows the attacker to ensure $M_S > M_R$ by jamming fewer wireless channels simultaneously. For example, the attacker may only need to eliminate legitimate APs by jamming channel 1 on the 2.4 GHz band using a single jammer (20% APs occupy Channel 1 in our collections).

C. Evaluation of the Countermeasure

We also evaluate the effectiveness of the aforementioned Countermeasure. We aim to evaluate the threshold ϵ so that one can identify legitimate APs by measuring the distribution of $\triangle\lambda_d$ of each detected AP.

We still use a Raspberry Pi 4B to extract CSI measurements. We first evaluate how the distance between Tx and Rx influences the distribution of $\triangle\lambda_d$. We set an AP that broadcasts one beacon frame every 100 milliseconds on Channel 11. In the meantime, the Raspberry Pi detects the beacon frames at different locations. Assume that d is the distance between two devices, where $d = 1, 5, 10, \ldots, 50$ meters. For each experiment, we take 5 seconds to collect 50 CSI measurements. Note that a mobile device takes 5 seconds to conduct a Wi-Fi scan. We then compute the distribution of $\triangle\lambda_d$ by calculating the mutual difference between CSI measurements. As a result, we observe that the variance of $\triangle\lambda_d$ is greater than 0.2 when $d > 10$ meters and is less than 0.2 when $d < 10$ meters. For example, the variance of $\triangle\lambda_d$ is less than 0.1 when d is around 1 m. Assume that the Wi-Fi spoofer is inside the same vehicle as the victim device and the distance between two devices is less than 10 meters. We take $\epsilon = 0.2$.

To further validate the threshold $\epsilon = 0.2$, we perform another experiment in real-world scenarios. We place one ESP8266 chipset inside the vehicle as the spoofer. It broadcasts crafted localization signals to mimic multiple falsified APs on channel 11. A Raspberry Pi inside the vehicle extracts CSI measurements from the ESP8266 chipset. It also collects CSI measurements from legitimate APs operating on channel 11. We then drive this vehicle for 2 miles around the campus where we conduct this research. As a result, we collect over 3,000 CSI measurements from the ESP8266 chipset and 156 legitimate APs. We then compute the variance of $\triangle\lambda_d$ of each AP. The variance of $\triangle\lambda_d$ of Wi-Fi signals from the ESP8266 chipset is less than 0.1 as the distance between two devices is about 1 m. 96 out of 156 legitimate APs are identified as their variances of $\triangle\lambda_d$ are greater than 0.2. The reason for the rest APs is that we did not collect sufficient CSI measurements from them. Note that the countermeasure is effective if more than two legitimate APs are identified.

VIII. Ethical Disclosure

Since the LLT inference attack exploits a design bug inside the Geolocation APIs from Google and Mozilla, we responsibly disclosed our attacks to them through their security disclosure portals in July 2022. Mozilla confirmed its vulnerability to the LLT inference attack on August 18. But they determined this vulnerability as low risk and decided to close it without fixing it. Google did not respond to the vulnerability report before the camera-ready deadline.

We perform the experiments in a confined way that does not affect other legitimate users. Specifically, we perform the spoofing attacks in a virtual environment with simulated victim users instead of real-world users. For the LLT inference attack, we only passively examine the geographic locations of APs in the location databases but never used any active method to profile the users/locations that use/host APs. All Wi-Fi MAC addresses are collected from publicly accessible Wi-Fi channels rather than any private networks. The number of API requests that we send to Google is within the normal range allowed by Google. We did not cause any levels of denial-of-service to a Google server, nor did we attempt to modify the LLT maintained by Google.

IX. Conclusion

In this paper, we demonstrate the LLT inference attack that can find out the precise locations of Wi-Fi APs, and we show that this attack enables an attacker to impact user privacy from various perspectives significantly and to successfully spoof the Google Wi-Fi positioning system in urban areas with a massive number of legitimate APs. We evaluate the effectiveness of the discovered location spoofing attack against the traditional location spoofing attack via experiments. We discuss the potential countermeasures to mitigate location spoofing attacks. We show that our countermeasures effectively identify legitimate APs using a single off-the-shelf Wi-Fi chipset without user interaction. The user can then verify his/her current location by making a Geolocation API request with these legitimate APs.

Acknowledgments

An abridged version of this paper appeared in the Proc. of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS), 2022 [18].

REFERENCES

[1] IEEE Standard for Information technology – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput, 2009.

[2] "Open source software-defined GPS signal simulator," 2018. [Online]. Available: https://github.com/osqzss/gps-sdr-sim

[3] "WALB (Wireless Attack Launch Box)," 2018. [Online]. Available: https://github.com/crescentvenus/WALB

[4] Combain Mobile AB, "Combain - Locate everything Everywhere," 2022. [Online]. Available: https://combain.com/

[5] G. Aissou, S. Benouadah, H. El Alami, and N. Kaabouch, "Instance-based supervised machine learning models for detecting GPS spoofing attacks on UAS," in *Proc. IEEE 12th Annu. Comput. Commun. Workshop Conf.*, 2022, pp. 0208–0214.

[6] D. M. Akos, "Who's afraid of the spoofer? GPS/GNSS spoofing detection via automatic gain control (AGC)," *Annu. Navigation*, vol. 59, pp. 281–290, 2012.

[7] F. Alrefaei, A. Alzahrani, H. Song, and S. Alrefaei, "A survey on the jamming and spoofing attacks on the unmanned aerial vehicle networks," in *Proc. IEEE Int. IOT, Electron. Mechatronics Conf.*, 2022, pp. 1–7.

[8] C. Arackaparambil, S. Bratus, A. Shubina, and D. Kotz, "On the reliability of wireless fingerprinting using clock skews," in *Proc. 3rd ACM Conf. Wirel. Netw. Secur.*, 2010, pp. 169–174.

[9] J. Armstrong, "Analysis of new and existing methods of reducing intercarrier interference due to carrier frequency offset in OFDM," *IEEE Trans. Commun.*, vol. 47, no. 3, pp. 365–369, Mar. 1999.

[10] H. Bolcskei, "Blind estimation of symbol timing and carrier frequency offset in wireless OFDM systems," *IEEE Trans. Commun.*, vol. 49, no. 6, pp. 988–999, Jun. 2001.

[11] A. Broumandan, A. Jafarnia-Jahromi, V. Dehghanian, J. Nielsen, and G. Lachapelle, "GNSS spoofing detection in handheld receivers based on signal spatial correlation," in *Proc. IEEE Position Location Navigation Symp.*, 2012, pp. 479–487.

[12] Y. Chen, W. Trappe, and R. P. Martin, "Detecting and localizing wireless spoofing attacks," in *Proc. IEEE 4th Annu. Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw.*, 2007, pp. 193–202.

[13] S. Dasgupta, M. Rahman, M. Islam, and M. Chowdhury, "A sensor fusion-based GNSS spoofing attack detection framework for autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 23559–23572, Dec. 2022.

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discov. Data Mining*, 1996, pp. 226–231.

[15] S. Filippou et al., "A machine learning approach for detecting GPS location spoofing attacks in autonomous vehicles," in *Proc. IEEE 97th Veh. Technol. Conf.*, 2023, pp. 1–7.

[16] D. Glasgow, ""Google maps updates to get you through the holidays"," 2020. [Online]. Available: https://blog.google/products/maps/google-maps-updates-get-you-through-holidays/

[17] J. Gu, J. Wang, Z. Yu, and K. Shen, "Walls have ears: Traffic-based side-channel attack in video streaming," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1538–1546.

[18] X. Han, J. Xiong, W. Shen, Z. Lu, and Y. Liu, "Location heartbleeding: The rise of Wi-Fi spoofing attack via geolocation API," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2022, pp. 1383–1397.

[19] A. Harvey, "Skylift: Wi-Fi geolocation spoofing with the ESP8266," 2016. [Online]. Available: https://github.com/adamhrv/skylift

[20] T. M. Hoang, T. van Chien, T. van Luong, S. Chatzinotas, B. Ottersten, and L. Hanzo, "Detection of spoofing attacks in aeronautical ad-hoc networks using deep autoencoders," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 1010–1023, 2022.

[21] J. Hua, H. Sun, Z. Shen, Z. Qian, and S. Zhong, "Accurate and efficient wireless device fingerprinting using channel state information," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1700–1708.

[22] T. E. Humphreys, "Detection strategy for cryptographic GNSS anti-spoofing," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 49, no. 2, pp. 1073–1090, Apr. 2013.

[23] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," *IEEE Trans. Mobile Comput.*, vol. 9, no. 3, pp. 449–462, Mar. 2010.

[24] K. Jansen, M. Schäfer, D. Moser, V. Lenders, C. Pöpper, and J. Schmitt, "Crowd-GPS-Sec: Leveraging crowdsourcing to detect and localize GPS spoofing attacks," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 1018–1031.

[25] K. Jansen, N. O. Tippenhauer, and C. Pöpper, "Multi-receiver GPS spoofing detection: Error models and realization," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 237–250.

[26] P. Jiang, H. Wu, Y. Zhao, D. Zhao, and C. Xin, "SEEK: Detecting GPS spoofing via a sequential dashcam-based vehicle localization framework," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2023, pp. 71–80.

[27] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 93–108, Second Quarter 2005.

[28] F. Lanze, A. Panchenko, B. Braatz, and T. Engel, "Letting the puss in boots sweat: Detecting fake access points using dependency of clock skews on temperature," in *Proc. 9th ACM Symp. Inf. Comput. Commun. Secur.*, 2014, pp. 3–14.

[29] F. Lanze, A. Panchenko, B. Braatz, and A. Zinnen, "Clock skew based remote device fingerprinting demystified," in *Proc. IEEE Glob. Commun. Conf.*, 2012, pp. 813–819.

[30] H. Li, Z. Xu, H. Zhu, D. Ma, S. Li, and K. Xing, "Demographics inference through Wi-Fi network traffic analysis," in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, 2016.

[31] S. Liu et al., "Stars can tell: A robust method to defend against GPS spoofing attacks using off-the-shelf chipset," in *Proc. 30th USENIX Secur. Symp.*, 2021.

[32] Google LLC, "Geolocation API: Usage and billing," 2022. [Online]. Available: https://developers.google.com/maps/documentation/geolocation/usage-and-billing

[33] Google LLC, "Wi-Fi scanning overview," 2022. [Online]. Available: https://developer.android.com/guide/topics/connectivity/wifi-scan

[34] Unwired Labs (India) Pvt. Ltd., ""Unwired labs location API," 2022. [Online]. Available: https://unwiredlabs.com/

[35] S. Markgraf, "osmo-fl2k: Using cheap USB 3.0 VGA adapters as SDR transmitter," 2015. [Online]. Available: https://osmocom.org/projects/osmo-fl2k/wiki/Osmo-fl2k

[36] K. Merry and P. Bettinger, "Smartphone GPS accuracy study in an urban environment," *PLoS One*, vol. 47, 2019, Art. no. e0219890.

[37] Mozilla, ""Mozilla location service"," 2022. [Online]. Available: https://location.services.mozilla.com/

[38] S. Narain, A. Ranganathan, and G. Noubir, "Security of GPS/INS based on-road location tracking systems," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 587–601.

[39] P. Olson, "Hacking A phone's GPS may have just got easier," 2015. [Online]. Available: https://www.forbes.com/sites/parmyolson/2015/08/07/gps-spoofing-hackers-defcon/?sh=e73fe954efbf

[40] C. Paget, "Practical cellphone spying," *Def Con*, 18, 2010.

[41] C. Pöpper, N. O. Tippenhauer, B. Danev, and S. Capkun, "Investigation of signal and message manipulations on the wireless channel," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2011, pp. 40–59.

[42] M. L. Psiaki, S. P. Powell, and B. W. O'Hanlon, "GNSS spoofing detection using high-frequency antenna motion and carrier-phase data," in *Proc. 26th Int. Tech. Meeting Satell. DiVis. Inst. Navigation*, 2013.

[43] Inc Qualcomm Technologies, "Skyhook | Location technology provider," 2022. [Online]. Available: https://www.skyhook.com/

[44] A. Ranganathan, H. Ólafsdóttir, and S. Capkun, "SPREE: A spoofing resistant GPS receiver," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016.

[45] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *Proc. 26th USENIX Secur. Symp.*, 2017.

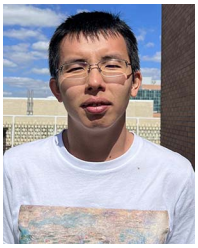
[46] N. Tadayon, M. T. Rahman, S. Han, S. Valaee, and W. Yu, "Decimeter ranging with channel state information," *IEEE Trans. Wirel. Commun.*, vol. 18, no. 7, pp. 3453–3468, Jul. 2019.

[47] N. O. Tippenhauer, K. B. Rasmussen, C. Pöpper, and S. Capkun, "Attacks on public WLAN-Based positioning systems," in *Proc. 7th Int. Conf. Mobile Syst. Appl. Serv.*, 2009.

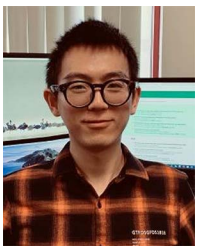
[48] M. Vanhoef and F. Piessens, "Advanced Wi-Fi attacks using commodity hardware," in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, 2014, pp. 256–265.

[49] J. A. Volpe, "Vulnerability assessment of the transportation infrastructure relying on global positioning system," 2001. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/8435

[50] C. Wang, C. Wang, Y. Chen, L. Xie, and S. Lu, "Smartphone privacy leakage of social relationships and demographics from surrounding access points," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 678–688.
[51] F. Wang, Y. Hong, and X. Ban, "Infrastructure-enabled GPS spoofing detection and correction," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 13878–13892, Dec. 2023.
[52] K. Wesson, M. Rothlisberger, and T. Humphreys, "Practical cryptographic civil GPS signal authentication," *NAVIGATION J. Inst. Navigation*, 2012.
[53] K. D. Wesson, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "An evaluation of the vestigial signal defense for civil GPS anti-spoofing," in *Proc. 24th Int. Tech. Meeting Satell. DiVis. Inst. Navigation*, 2011.
[54] WiGLE.NET, ""WiGLE: Wireless network mapping"," 2022. [Online]. Available: https://wigle.net/index
[55] Y. Xie, Z. Li, and M. Li, "Precise power delay profiling with commodity Wi-Fi," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 53–64.
[56] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *Proc. 6th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2005, pp. 46–57.
[57] N. Xue, L. Niu, X. Hong, Z. Li, L. Hoffaeller, and C. Pöpper, "DeepSIM: GPS spoofing detection on UAVs using satellite imagery matching," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2020, pp. 304–319.
[58] Z. Yang et al., "Anomaly detection against GPS spoofing attacks on connected and autonomous vehicles using learning from demonstration," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 9, pp. 9462–9475, Sep. 2023.
[59] K. C. Zeng et al., "All your GPS are belong to us: Towards stealthy manipulation of road navigation systems," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1527–1544.

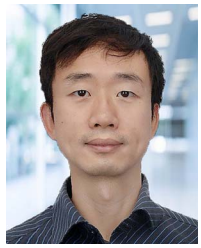


Xiao Han received the BE degree in electrical engineering from Northwest A & F University, in 2016, and the MS degree in electrical and computer engineering from the University of Arizona, in 2019. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, University of South Florida. His research interests include Wireless networks and network security.



Junjie Xiong received the BS degree in computer science from the Beijing University of Posts and Telecommunications, China, in 2012, and the MS degree in computer science from the Nanjing University of Posts and Telecommunications, China, in 2019. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, University of South Florida, US. His research interests include Wireless networks and network security.

Wenbo Shen received the PhD degree from the Computer Science Department, North Carolina State University, in 2015. He is currently a ZJU 100-Young professor with Zhejiang University, China. His research interests include system security and software security, including container security, OS kernel security, and program analysis using LLVM/clang.



Mingkui Wei is an associate professor with the Cyber Security Engineering Department, George Mason University. His current research interests include computer network and web application security, where he endeavors to identify, evaluate, and combat security threats among various web pages and websites. In the past, he has focused on cyber-physical system security and reliability that cover Smart Grid and Intelligent Transportation Systems. His research works have been published in top-notch venues including IEEE INFOCOM, ACM CCS, Usenix Security, etc.

Shangqing Zhao received the PhD degree in electrical engineering from the University of South Florida, Tampa, Florida, in 2021. He is currently an assistant professor with the School of Computer Science, University of Oklahoma (Tulsa campus). His research primarily focuses on novel mobile system design, mobile and network security. His recent research is equally focused on machine learning for network and security applications.

Zhuo Lu received the PhD degree from North Carolina State University, in 2013. He is an associate professor with the Department of Electrical Engineering, University of South Florida. He is also affiliated with the Florida Center for Cybersecurity and by courtesy with the Department of Computer Science and Engineering. His research has mainly focused on theoretical and system perspectives on communication, network, and security. His recent research is equally focused on machine learning and AI perspectives on networking and security. He received the NSF CISE CRII award in 2016, the Best Paper Award from IEEE GlobalSIP in 2019, and the NSF CAREER award in 2021.

Yao Liu received the PhD degree in computer science from North Carolina State University, in 2012. She is an associate professor with the Department of Computer Science and Engineering, University of South Florida. Her research is related to computer and network security, with an emphasis on designing and implementing defense approaches that protect mobile, network, and computer technologies from being undermined by adversaries. Her research interests include security applications for cyber-physical systems, Internet of Things, and machine learning. She was an NSF CAREER Award recipient in 2016. She also received the ACM CCS Test-of-Time Award by ACM SIGSAC in 2019.