# Location Heartbleeding: The Rise of Wi-Fi Spoofing Attack Via Geolocation API

Xiao Han
University of South Florida
Tampa, FL, USA
xiaoh@usf.edu

Junjie Xiong
University of South Florida
Tampa, FL, USA
junjiexiong@usf.edu

Wenbo Shen
Zhejiang University
Hangzhou, Zhejiang, China
shenwenbo@zju.edu.cn

Zhuo Lu
University of South Florida
Tampa, FL, USA
zhuolu@usf.edu

Yao Liu
University of South Florida
Tampa, FL, USA
yliu21@usf.edu

## 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.

## CCS CONCEPTS

• **Security and privacy** → *Mobile and wireless security*; *Web application security*; • **Networks** → *Wireless access points, base stations and infrastructure.*

## KEYWORDS

Wi-Fi Localization; Localization Attacks; Geolocation APIs

**ACM Reference Format:**
Xiao Han, Junjie Xiong, Wenbo Shen, Zhuo Lu, and Yao Liu. 2022. Location Heartbleeding: The Rise of Wi-Fi Spoofing Attack Via Geolocation API. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560623


*CCS '22, November 7–11, 2022, Los Angeles, CA, USA.*

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560623

## 1 INTRODUCTION

Wi-Fi positioning systems have been commercialized and widely used as a complement to conventional GPS-based positioning systems. 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 where there are dozens of Wi-Fi networks" [18], and [52] mentions that spoofing attacks failed to spoof a victim 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. Tippenhauer *el al.* [52] proposes to remove Wi-Fi signals from legitimate Wi-Fi APs by using 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 [54, 63].

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?

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 for over 1 billion people worldwide [15]. 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 physical-layer 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. It enables the attacker to obtain the precise locations of APs in LLT, no matter where these APs are located. We thoroughly validate this attack and point out the restrictions on conducting this attack.
- 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 the 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 location 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.
- In addition to the location spoofing attack, we discover that the LLT inference attack can raise other severe privacy concerns. For example, the precise location of an AP enables an attacker to know the exact address of the apartment, house, building, office, etc that hosts this AP. As a result, the attacker is able to infer sensitive information (e.g., precise locations, video content that they watch, and demographics) through Wi-Fi traffic analysis targeting a specific household. Besides the Google location service, we further study the vulnerability of other location services (e.g., Mozilla, Skyhook Wireless, and WiGLE) and identify that the Mozilla location service is vulnerable to the LLT inference attack.
- We evaluate the performance of the location spoofing attack using public APs collected in the city where we conducted this research. The result shows that an attacker can successfully fool a victim device with a success rate of 0.99 (i.e. 693 out of 696 trials) compared to that of 0.5 achieved by the traditional spoofing attack without jamming. We also discuss the potential countermeasures to mitigate the discovered location spoofing and the LLT inference attacks.

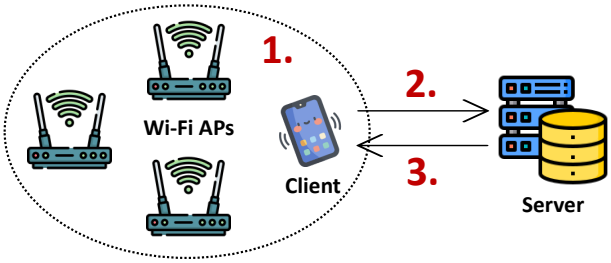



**Figure 1: Localization process of Wi-Fi positioning system: 1. The client device detects Wi-Fi APs in its reception range. 2. The client device queries the LLT server. 3. The server returns a position estimate based on the received Wi-Fi information.**

## 2 BACKGROUND

In this section, we introduce the preliminaries of Wi-Fi positioning systems, then the Google Geolocation API, and finally present an overview of our work.

### 2.1 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 a 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 Figure 1. In step 1, a mobile device scans all 802.11a/b/g channels for visible Wi-Fi APs. Meanwhile, it records their MAC addresses and corresponding signal strengths once detects these APs. In step 2, the mobile device queries the Google LLT server with the recorded MAC addresses and corresponding 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.

## 2.2 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 mobile 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 Wi-Fi APs, the request payload `wifiAccessPoints` must contain at least two Wi-Fi access point objects. For each Wi-Fi AP object, 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 and represents the radius of a circle around the given `location`. 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 Wi-Fi MAC addresses would result in an error indicating *no results were found even the request was valid*. In most cases, Google Geolocation API is adopted to provide accurate position estimates for web services, such as restaurant recommendations.

## 2.3 Overview of the Attack

In this section, we present an overview of our attacks. Our work can be categorized as follows:

First, we demonstrate *the LLT inference attack* in Section 3, which is capable of discovering the geographic location of each AP available in LLT from the Google location database. We validate this attack and present the restrictions on launching it. In Section 4, we show the privacy concerns raised by the LLT inference attack.

Second, in Section 5, we show how the revealed locations of APs in LLT enable an attacker to launch a successful location spoofing attack in dense urban areas without jamming. Our findings are summarized into four criteria. In Section 6, we evaluate the effectiveness of our spoofing attack under different scenarios. We also show the weakness of the traditional spoofing attacks even under optimal conditions. Finally, we discuss the potential countermeasures mitigate our spoofing attack in Section 7.

# 3 LLT INFERENCE ATTACK

As mentioned earlier, we discover that an attacker can use the LLT inference attack to reveal the geographic locations of APs in LLT from the Google location database. We thoroughly discuss the privacy threat raised by this technique. Eventually, it enables an attacker to construct the location spoofing attack without relying on jamming techniques 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 simply use an AP that its 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 the detected 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 3.3), these position estimates can leak the LLT to the attacker. 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 in the fields of `macAddress` and `signalStrength` for each of the two APs. Other fields such as `channel` and `signalToNoiseRatio` are optional and they do not impact on the localization results. The attacker can just rule out these fields.

## 3.1 Range-free Mode

We start by demonstrating the LLT inference attack using the range-free mode as shown in Figure 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

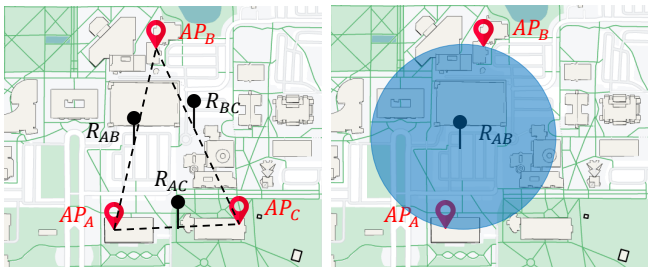
```
{
  "considerIp": "false",
  "wifiAccessPoints": [
    {
      "macAddress": "00:24:7C:XX:XX:99"
    },
    {
      "macAddress": "00:24:7C:XX:XX:0A"
    }
  ]
}
```


(a) Range-free mode


```
{
  "considerIp": "false",
  "wifiAccessPoints": [
    {
      "macAddress": "00:24:7C:XX:XX:99",
      "signalStrength": -45
    },
    {
      "macAddress": "00:24:7C:XX:XX:0A",
      "signalStrength": -55
    }
  ]
}
```


(b) Range-based mode

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




(a) (b)

**Figure 3: LLT inference attack: (a) 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 and (b) $AP_A$ and $AP_B$ lie on the circle around the position estimate $R_{AB}$ given the accuracy field.**

the estimated distances between APs and a mobile device in the 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 Figure 3(a), 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 $AP_C$ respectively, $Y_A$, $Y_B$, and $Y_C$ represent the longitude of $AP_A$, $AP_B$, and $AP_C$ respectively, $X_{AB}$, $X_{BC}$, and $X_{AC}$ is the latitude of $R_{AB}$, $R_{BC}$, and $R_{AC}$ respectively, and $Y_{AB}$, $Y_{BC}$, 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 directly solved from Equation (1). They form the geographic locations of these APs residing in LLT.

**Validation.** We validate the conjecture that the above position estimate is the middle point leveraging the `accuracy` field of each Geolocation API response. For each API request, in addition to the position estimate, Google also returns an `accuracy` value that describes how accurate the localization is. The physical location of the mobile device falls in a circle that centers at the position estimate with a radius of the accuracy value. Assume that $AP_A$ and $AP_B$ indeed lie on the circle and the distance between $AP_A$ and $AP_B$ is the diameter of this circle as seen in Figure 3(b). As a result, the returned `accuracy` value must be equal to half of the distance between the inferred geographic locations of $AP_A$ and $AP_B$. To validate our conjecture, we perform a trial of experiments using 1,590 public APs, which are collected while driving 1.6 miles along a route starting from location $L_A$ where we detect $AP_A$. We then make API requests with pairwise APs between each of the 1,590 APs and $AP_A$ using the range-free mode and obtain the position estimate and the `accuracy` value of each API request. The geographic location of $AP_A$ in LLT is calculated using Equation (1).

Our intuition is that, given the known location of $AP_A$ and the position estimates by making API requests with pairwise APs between 1,590 APs and $AP_A$, the distance between the location of $AP_A$ and each position estimate must be equal to the corresponding `accuracy` value if the position estimate is indeed the midpoint between two APs according to our conjecture. We then calculate such a distance and compare it with the `accuracy` value. We observe from experimental results that the `accuracy` value is almost equal to the distance between each position estimate and the location of $AP_A$ with negligible error. For example, the maximum difference that we have measured between the distance and the `accuracy` value is only about 0.95 meters even the measured distance is over 1,000 meters. Such a result demonstrates that the position estimate is the middle point on the line between two APs by making an API request with 2 APs using the range-free mode. Therefore, we conclude that an attacker is capable of inferring the geographic locations of APs in LLT using Equation (1).

**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 for an API request with these APs, 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 by solving Equation (1) before October 2021. For example, we have obtained the locations of $AP_A$ and $AP_B$ in LLT detected from location $L_A$ and $L_B$ before October 2021, respectively. 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.

## 3.2 Range-based Mode

Although the Google Geolocation API directly returns the geographic 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 Figure 2(b). For each AP in this API request, the attacker also includes a field of `signalStrength` that 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 $AP_B$. We further make use of 17 unique APs with the SSID (Service Set Identifier) of "McDonalds Free WiFi" from 10 different McDonald's restaurants in the city where we perform this research. For each AP, we make an API request with $AP_A$, while assigning the fields of `signalStrength` for this AP and $AP_A$ with -45 and -55, respectively. As a result, we successfully discover the geographic locations of all APs at 10 different McDonald's restaurants. By contrast, when we assign the field of `signalStrength` for each AP and $AP_A$ with -55 and -45, respectively, all API requests return the position estimates that are the same as the location of $AP_A$.

## 3.3 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.

*3.3.1 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 the Geolocation API requests per day. Nevertheless, each API key is limited to 100 Geolocation API requests per second.

*3.3.2 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 the revealed location of $AP_A$. 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 again make use of 1,590 public APs in Section 3.1. 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 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 AP 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. Note that the `accuracy` of 180 meters is an empirical observation from Google according to our extensive tests. We never obtain an `accuracy` that is greater than 180 meters by making an API request with two APs. We determine the distance in meters

between two geographic locations using the haversine formula, which is used to determine the great-circle distance between two points on a sphere given their latitudes and longitudes [60].

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 [59]. 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.

*3.3.3 Restriction on RSS.* The `signalStrength` is used to indicate the amplitude of radio signal measured by a wireless receiver like a mobile device. For example, an Atheros Wi-Fi chipset can return an RSS value ranging from 0 to -127 [61]. We aim to examine if all available RSS values are suitable for the LLT inference attack under the range-based mode. According to our testing results, we notice that, by making an API request with pairwise APs using the range-based mode, Google returns a valid response when both `signalStrength` values are over -100 but less than -35. Otherwise, it returns an error indicating no results were found.

## 4 IMPLICATION OF THE LLT INFERENCE ATTACK ON USER PRIVACY

Our results show the effectiveness of the LLT inference attack in discovering the geographic locations of APs in LLT by making API requests with two APs, especially without the restriction on the maximum distance between these APs. In this section, we discuss the privacy issues that may be raised by the LLT inference attack.

### 4.1 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.

**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 using the LLT inference attack and then obtain the street address of the AP by feeding the inferred location into the reverse-geocoding API.

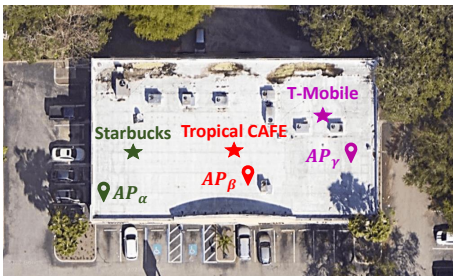



**Figure 4: The geographic locations of $AP_\alpha$, $AP_\beta$, and $AP_\gamma$ from Starbucks, Tropical Smoothie Cafe, and T-Mobile in the Google location database, respectively.**

To avoid the 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 geographic 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 $AP_\alpha$, $AP_\beta$, and $AP_\gamma$ 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 accurately differentiate these stores using the revealed locations of their respective APs in LLT as shown in Figure 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., precise locations, video content that they watch, and demographics) through Wi-Fi traffic analysis [6, 13, 16, 30, 47, 48, 56]. 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 households nearby. Without the LLT inference attack, an attacker who does not know the street address of this AP can only associate the inferred sensitive information to all households in the same community. Otherwise, to localize this AP, the attacker may have to utilize specialized hardware (e.g., a directional antenna) 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 correct street address of each AP given its Wi-Fi MAC address with negligible costs.

## 4.2 Privacy Threat Example II: Monitoring Relocation and Travel

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 [18]. 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 by [18], we infer that it was moved from California to Germany by its owner. Then Google updated its location in LLT to Germany.

Similarly, 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 an AP was moved to Malaysia. Because Wi-Fi MAC address is a global unique identifier assigned to each AP, we infer that these APs were potentially relocated with their owners during the last two months.

We further performed an experiment using the AP owned by one of the authors to test how long it takes Google to update the location of an AP. We first obtained the geographic location of this AP using the LLT inference attack in March 2022. We then brought it to a building 6.1 miles away from the original place. Next, we monitored the location of this AP in LLT each day and found that Google updates the location of the AP in six days.

According to reports, about 24% of U.S. adults moved within the country in the past five years [12]. In 2021, new remote work opportunities further enabled an estimated number of 14-23 million Americans to relocate [40]. In addition, many people relocate while carrying the same APs they have already configured with the SSIDs and the passwords. Thus, the LLT inference attack can impose a practical threat against users' location privacy on a large scale.

## 4.3 Privacy Threat Example III: Getting the Location without Requesting Permission

It has been shown that an application installed on a smart device running Android 9 or lower can obtain the MAC address of a Wi-Fi AP it associates with without requesting any permission [46, 66]. Thus, even if a malicious app is denied permission to access to location (e.g., GPS, Wi-Fi positioning system, or Cellular), it can still use the LLT inference attack to discover the precise location of a smart device using the Wi-Fi MAC address it connects to. However, for Android 10 or higher, applications must have the `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION` permissions to access the MAC address of a Wi-Fi AP it connects to [33]. Therefore, this location privacy leakage is prevalent for smart devices running Android 9 or lower.

## 4.4 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 also vulnerable to the LLT inference attack.

*4.4.1 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 [59].

WiGLE is the only Wi-Fi location database that allows a single MAC address lookup. By providing the MAC address of one Wi-Fi AP, WiGLE API 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 4.2, 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. Thence, 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 provided by WiGLE for an AP is extremely low. For 137 public APs available in the Google location database mentioned in Section 4.1, we obtain the geographic 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 $AP_\alpha$, $AP_\beta$, and $AP_\gamma$ 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 Figure 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.

*4.4.2 Mozilla Location Service.* We identify that the Mozilla location service is 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 has recorded over 2.3 billion Wi-Fi APs worldwide in July 2022 [38]. 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 geographic locations of APs in the Mozilla location database by leveraging the position estimates returned from API requests with pairwise APs.

**Validation.** We again validate this vulnerability using our conjecture about the `accuracy` value as discussed in Section 3.1. That is, assume Mozilla indeed returns the middle point between the locations of two APs as the position estimate by making an API request with these APs. The locations of these APs must lie on the circle that centers at the position estimate, and the `accuracy` value as the radius must be equal to the distance between the position estimate and the inferred location of each AP. Let $AP_A$, $AP_B$, and $AP_C$ indicate three APs available in the Mozilla location database, 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, Mozilla returns the position estimate $R_{AB}$, $R_{BC}$, and $R_{AC}$, respectively. We then calculate the locations $L_A$, $L_B$, and $L_C$ of $AP_A$, $AP_B$, and $AP_C$ in the database by solving Equation (1), respectively. The distance between $L_A$ and $R_{AB}$ is about 233.4 meters, and the corresponding `accuracy` value obtained from an API request with $AP_A$ and $AP_B$ is about 232.6 meters. The difference is only about 0.8 meters. Similarly, the distance between $L_A$ and $R_{AC}$ is about 87.2 meters, and the `accuracy` value is about 86.9 meters resulting in a difference of 0.3 meters. We further validate our conjecture by making use of 216 APs available in the Mozilla location database. All these APs are collected on campus. For each AP, we make an API request with $AP_A$ using the range-free mode and obtain the position estimate. We then calculate the distance between each position estimate and $L_A$ and compare it with the corresponding `accuracy` value. We observe that the `accuracy` value is always equal to the distance with negligible error. The maximum difference that we have measured is only about 1.5 meters. Thus, we conclude that the Mozilla location service is also vulnerable to the LLT inference attack.

**Restrictions.** According to our experimental results, we notice that the Geolocation API from Mozilla returns a valid response when the distance between the geographic locations of two APs is less than 500 meters. Otherwise, an error is returned. The usage of the Geolocation API is free of charge. However, the Mozilla location service sets a daily limit of 100,000 requests for each valid API key.

*4.4.3 Other Location Services.* We also attempt to investigate other location services, like Combain Positioning Service [4], Unwired Labs [34], and Skyhook Wireless [44]. 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 wide area positioning 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 documentations, 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.

# 5 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 dozens of legitimate APs are present. Our findings are summarized into four criteria, satisfying 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, an 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 a better location accuracy.

## 5.1 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 dense 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. However, it is suspicious to the victim device if it cannot detect any Wi-Fi APs on a most occupied channel (e.g., Channel 1, 6, and 11), especially in dense urban areas.

**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 80MHz 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 1GHz bandwidth nowadays, including channels on 2.4GHz and 5GHz bands [1]. Jamming all channels concurrently can cause a non-trivial cost for the attacker even if the attacker adopts reactive jammers [42, 53]. 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 existing Wi-Fi location spoofing attacks. Unfortunately, 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 all possible Wi-Fi channels.

## 5.2 Criterion I

*Assume Google receives the aggregated 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 to determine the position estimate when it receives mixed Wi-Fi information from different locations. We further 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 slightly shifted away from location $L_A$ (e.g., less than 50 meters) rather than switching to $L_B$. 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 smartphone 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 localization 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 the Nexus 6 smartphone is spoofed to location $L_B$. The distance between location $L_B$ and the open area is over 1,000 meters.

## 5.3 Criterion II

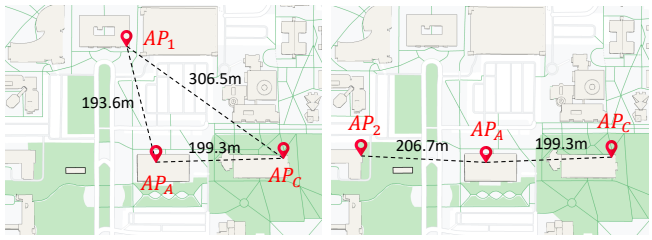
*Assume Google receives mixed Wi-Fi information from different locations. In order to localize the client device, Google first clusters APs using a radius of 200 meters and then determines the position estimate based on a cluster with the maximum number of APs.*

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 still feasible to fool the victim 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?

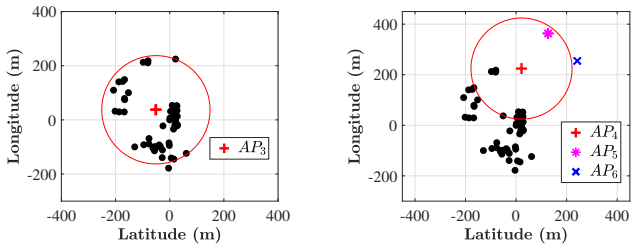
We address this question leveraging Criterion I. Specifically, given $M_R$ legitimate APs from location $L_R$ and $M_S$ falsified APs from location $L_S$, where $M_R = M_S$, the Geolocation API returns a position estimate between the two locations by making an API request with $M_R$ and $M_S$ using the 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 $L_S$.

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 Figure 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, and 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. $i = 1, 2, ..., 1, 590$. 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:




(a) (b)

**Figure 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 $AP_i$.**




(a) (b)

**Figure 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.**

(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 Figure 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 Figure 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 Figure 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 our observation, we perform a trial of experiments using dozens of APs, shown in Figure 6. Our strategy still leverages 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, Figure 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 $AP_3$ and has a radius of 200 meters.

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 Figure 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 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 the Google Geolocation API. Ultimately, in Section 6, we show that the radius of 200 meters is sufficient enough for an attacker to maximize its success rate of location spoofing attacks in dense urban areas without jamming.

## 5.4 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 Geolocation API returns an error if the request consists of multiple APs located at one unique geographic location since October 2021. However, the API used to treat such a request as valid and returned the location of the APs 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.

### 5.5 Criterion IV

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

### 5.6 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. The goal of the attacker is to spoof a victim device from its current 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 case that $M_R > M_S$, we argue that jamming attacks are still necessary to conduct a successful attack. Nevertheless, in Section 6.2, 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 6.2.

## 6 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.

### 6.1 Evaluation of the Traditional Attack

We start by evaluating the traditional Wi-Fi location spoofing attack demonstrated in [18, 52]. 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 $i = 1, 2, ..., n$. 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 $1 \le i \le n$.

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.

### 6.2 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 5.6.

Again, our goal is to fool the victim device from its current location around campus to a faraway location near the downtown area. We thus take APs detected from different routes close to the university 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 $M_S^i$. Consider 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 that $M_R > M_S$, 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 2.4GHz and 5GHz bands in most trials. Consequentially, the traditional attack has to jam at least 6 channels by using multiple jammers simultaneously. In comparison to the traditional attack, the one discovered in this paper allows the attacker to ensure $M_S > M_R$ by jamming fewer wireless channels at the same time. For example, the attacker may only need to eliminate legitimate APs by jamming channels 1 on the 2.4GHz band using a single jammer (20% APs occupy Channel 1 in our collections).

# 7 DISCUSSION AND COUNTERMEASURES

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 describe potential countermeasures to mitigate this attack.

## 7.1 Using Reference APs

A victim device detects mixed Wi-Fi information from two distinct locations for each Wi-Fi scan if an attacker is present. To detect the presence of an attack, instead of making an API request with detected 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 by the device from a reference location far away from its current location. Let $N_C$ and $N_V$ indicate the numbers of nearby and the reference APs in an API request, respectively, where $N_C = N_V + 1$. Note that the reference APs, $N_C$, and the corresponding location should be kept confidential.

Recall that when Google receives the Wi-Fi information from two locations in an API request, Google returns a position estimate that is the location with a larger number of APs. If there is no attack, the API request with nearby and reference APs will return a correct position estimate around the current location rather than the reference location because $N_C > N_V$. In the presence of the attack, as long as the number of falsified APs is equal to or larger than 2 and so does the number of legitimate APs, the API request will return a position estimate that is in the vicinity of the reference location. To explain the reason, we know that $N_C = N_V + 1 = n_1 + n_2$, where $n_1$ and $n_2$ are the numbers of detected legitimate and falsified APs from two distinct locations, respectively. We can derive that $n_1 = N_V + 1 - n_2 < N_V$ and $n_2 = N_V + 1 - n_1 < N_V$ if $n_1 \geq 2$ and $n_2 \geq 2$. Hence, the Google Geolocation API returns a position estimate around the reference location because $N_V$, $n_1$, and $n_2$ form this request and $N_V > n_1$ and $N_V > n_2$.

This defense method is simple and easy to implement, but it requires at least 2 legitimate and falsified APs (i.e., $n_1 \geq 2$ and $n_2 \geq 2$) and may generate a high false alarm rate because some APs from the nearby or reference set may not be recorded by the LLT.

## 7.2 Using Physical Layer Features

One key application of the location spoofing attack 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., [65]). An attacker may also want to subvert the Wi-Fi localization by launching the location spoofing attack discovered in this paper. In the context of car navigation, one may differentiate legitimate and falsified APs leveraging the physical-layer characteristics of Wi-Fi signals.

We assume that the spoofer is placed in the same car as the navigation system. Therefore, we may detect the attack by measuring the time-of-flight (ToF) and angle-of-arrival (AoA) of received Wi-Fi signals using channel state information. ToF and AoA have been extensively studied for indoor localization. For example, Tadayon *et al.* [50] explored the feasibility of decimeter ranging using channel state information with 20 MHZ bandwidth. Kotaru *et al.* [26] first demonstrated how to accurately compute AoA of multipath components using three antennas. The navigation system may measure ToF and AoA to identify the existence of the attack because the ToF and AoA measured from crafted signals emitted by the spoofer are more stable than those from legitimate APs.

Alternatively, it may be possible to detect the attack by examining the Doppler effect. In particular, the frequency of the Wi-Fi signals from a legitimate AP that received by the navigation system decreases as the car moves away from this AP, and increases as the car moves towards the AP. On the other hand, because the spoofer is placed within the same vehicle as the navigation system, such Doppler effect does not exist due to the lack of relative velocity between the spoofer and the navigation system. Thus, the navigation system may attempt to observe the Doppler effect to detect whether Wi-Fi signals are from the spoofer or legitimate APs. These defense methods assume the scenarios of car navigation and require additional hardware to measure the physical layer features like ToF, AoA, and the Doppler shift.

## 7.3 Mitigation of the LLT Inference Attack

The attacker leverages the LLT inference attack to enable the location spoofing attack in dense urban areas. Thus, mitigating the LLT inference attack can be helpful in dealing with the location spoofing attack. Google may mitigate the LLT inference attack by slightly randomizing the position estimate such that the position estimate is no longer equal to the location of a certain AP in an API request. In particular, for an API request with two APs using the range-based mode, a client gets the location estimate that is close to the AP with the larger value of `signalStrength` but not exact the same as the precise location of this AP in LLT. The drawback of this method is that it may be hard to determine appropriate randomization offset that can achieve both desired security and localization accuracy.

## 8 RELATED WORK

Location spoofing attacks targeting mobile navigation systems were first discussed in [55]. The authors pointed out that malicious interference to the civilian GPS signals can be a serious 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 [36, 41, 51]. Zeng *et al.* [65] 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 to deal with this attack is to use inertial sensors to keep track of a vehicle's movement patterns. However, Sashank *et al.* [39] revealed attacks against this approach. The attacker can fool the victim to drive on a route that yields inertial sensors readings similar to these from the original path. In general, both cryptographic and non-cryptographic countermeasures were proposed to address GPS spoofing attacks (e.g., [20, 27, 57], [5, 10, 22, 23, 43, 45, 58, 64]). However, these techniques do not target the aforementioned advanced attacks (i.e, [39, 65]) and some of them need modifications to existing GPS infrastructure. Liu *et al.* [31] presented a new defense method to localize a GPS spoofer by iteratively moving towards the incoming angle of fake GPS signals leveraging the receiver's rotation and significant signal attenuation caused by the physical blockage.

The Wi-Fi positioning system is a complement to GPS when GPS signals are weak in urban canyon environments [37]. According to our study, Google Maps shows the position estimate from GPS or Wi-Fi positioning systems according to the localization accuracy. 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. Therefore, to fool a victim in dense urban areas without raising alarms, it is equally important for an attacker to stealthily launch location 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.

The location spoofing attack against Wi-Fi positioning was first systematically investigated and discussed in [52]. Skylift, an online project, further demonstrates the possibility of launching this attack by using a low-cost Wi-Fi microchip ESP8266 [17]. Packetbridge also explores the vulnerability of Wi-Fi positioning systems by simulating network situations from different infrastructures [49]. Jamming is usually considered necessary to make these attacks successful. In this paper, we identify the LLT inference attack which provides the precise locations of APs in the location databases, and this information enables an attacker to discover the black-box localization algorithms used by the Google Wi-Fi positioning system and intelligently craft the location spoofing attack with a significantly reduced effort of jamming.

On the other hand, several countermeasures have been proposed to detect unauthorized Wi-Fi APs [11, 24, 28, 29, 35, 62]. Kohno *et al.* [25] first proposed the use of the clock skew of a computer on a network for the purpose of authenticating Wi-Fi APs. 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 Wi-Fi APs. On the wireless side, Jana *et al.* [21] 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 [7]. Alternatively, Hua *et al.* [19] 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 [8], [9]. 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.

## 9 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 fix. 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 location 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.

## 10 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 significantly impact on user privacy from various perspectives, and to successfully spoof the Google Wi-Fi positioning system in urban areas with a massive number of legitimate APs. We further evaluate the effectiveness of the discovered location spoofing attack against the traditional location spoofing attack via experiments.

## ACKNOWLEDGMENTS

The authors would like to thank all anonymous reviewers for their insightful comments. This research was supported by the National Science Foundation under grants CNS-1553304 and CNS-2044516. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of the NSF.

# 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. https://github.com/osqzss/gps-sdr-sim, 2018.

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

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

[5] Dennis M. Akos. Who's afraid of the spoofer? GPS/GNSS spoofing detection via automatic gain control (AGC). *Annual of Navigation*, 2012.

[6] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic. *arXiv preprint arXiv:1705.06805*, 2017.

[7] Chrisil Arackaparambil, Sergey Bratus, Anna Shubina, and David Kotz. On the reliability of wireless fingerprinting using clock skews. In *Proceedings of the Third ACM Conference on Wireless Network Security (WiSec)*. Association for Computing Machinery, 2010.

[8] Jean. Armstrong. Analysis of new and existing methods of reducing intercarrier interference due to carrier frequency offset in OFDM. *IEEE Transactions on Communications*, 1999.

[9] Helmut. Bolcskei. Blind estimation of symbol timing and carrier frequency offset in wireless OFDM systems. *IEEE Transactions on Communications*, 2001.

[10] Ali Broumandan, Ali Jafarnia-Jahromi, Vahid Dehghanian, John Nielsen, and Gérard Lachapelle. GNSS spoofing detection in handheld receivers based on signal spatial correlation. In *in Proceedings of the IEEE Position Location and Navigation Symposium (PLANS)*, 2012.

[11] Yingying Chen, Wade Trappe, and Richard P. Martin. Detecting and localizing wireless spoofing attacks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2007.

[12] D'vera Cohn. "About a fifth of U.S. adults moved due to COVID-19 or know someone who did". https://www.pewresearch.org/fact-tank/2020/07/06/about-a-fifth-of-u-s-adults-moved-due-to-covid-19-or-know-someone-who-did/, 2020.

[13] Bogdan Copos, Karl Levitt, Matt Bishop, and Jeff Rowe. Is anybody home? inferring activity from smart home network traffic. In *2016 IEEE Security and Privacy Workshops (SPW)*, 2016.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96. AAAI Press, 1996.

[15] Dane Glasgow. "Google Maps updates to get you through the holidays". https://blog.google/products/maps/google-maps-updates-get-you-through-holidays/, 2020.

[16] Jiaxi Gu, Jiliang Wang, Zhiwen Yu, and Kele Shen. Walls have ears: Traffic-based side-channel attack in video streaming. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[17] Adam Harvey. "Data Pools: Wi-Fi Geolocation Spoofing". https://ahprojects.com/datapools/, 2016.

[18] Adam Harvey. Skylift: Wi-Fi Geolocation Spoofing with the ESP8266. https://github.com/adamhrv/skylift, 2016.

[19] Jingyu Hua, Hongyi Sun, Zhenyu Shen, Zhiyun Qian, and Sheng Zhong. Accurate and efficient wireless device fingerprinting using channel state information. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[20] Todd E Humphreys. Detection strategy for cryptographic GNSS anti-spoofing. *IEEE Transactions on Aerospace and Electronic Systems*, 2013.

[21] Suman Jana and Sneha K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE Transactions on Mobile Computing*, 2010.

[22] Kai Jansen, Matthias Schäfer, Daniel Moser, Vincent Lenders, Christina Pöpper, and Jens Schmitt. Crowd-GPS-Sec: Leveraging Crowdsourcing to Detect and Localize GPS Spoofing Attacks. In *IEEE Symposium on Security and Privacy (SP '18)*, 2018.

[23] Kai Jansen, Nils Ole Tippenhauer, and Christina Pöpper. Multi-receiver gps spoofing detection: Error models and realization. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, 2016.

[24] Taebeom Kim, Haemin Park, Hyunchul Jung, and Heejo Lee. Online detection of fake access points using received signal strengths. In *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*, 2012.

[25] Tadayoshi. Kohno, Andre. Broido, and Kimberly. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2005.

[26] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. SpotFi: Decimeter level localization using WiFi. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[27] Markus G Kuhn. An asymmetric security mechanism for navigation signals. In *International workshop on information hiding*, 2004.

[28] Fabian Lanze, Andriy Panchenko, Benjamin Braatz, and Thomas Engel. Letting the puss in boots sweat: Detecting fake access points using dependency of clock skews on temperature. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*. Association for Computing Machinery, 2014.

[29] Fabian Lanze, Andriy Panchenko, Benjamin Braatz, and Andreas Zinnen. Clock skew based remote device fingerprinting demystified. In *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012.

[30] Huaxin Li, Zheyu Xu, Haojin Zhu, Di Ma, Shuai Li, and Kai Xing. Demographics inference through Wi-Fi network traffic analysis. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016.

[31] Shinan Liu, Xiang Cheng, Hanchao Yang, Yuanchao Shu, Xiaoran Weng, Ping Guo, Kexiong (Curtis) Zeng, Gang Wang, and Yaling Yang. Stars can tell: A robust method to defend against GPS spoofing attacks using off-the-shelf chipset. In *30th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2021.

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

[33] Google LLC. Privacy changes in Android 10. https://developer.android.com/about/versions/10/privacy/changes#proc-net-filesystem, 2022.

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

[35] Mahabub Hasan Mahalat, Shreya Saha, Anindan Mondal, and Bibhash Sen. A PUF based light weight protocol for secure WiFi authentication of IoT devices. In *2018 8th International Symposium on Embedded Computing and System Design (ISED)*, 2018.

[36] Steve Markgraf. osmo-fl2k: Using cheap USB 3.0 VGA adapters as SDR transmitter. https://osmocom.org/projects/osmo-fl2k/wiki/Osmo-fl2k, 2015.

[37] Krista Merry and Pete Bettinger. Smartphone GPS accuracy study in an urban environment. *PloS one*, 2019.

[38] Mozilla. "Mozilla Location Service". https://location.services.mozilla.com/, 2022.

[39] Sashank Narain, Aanjhan Ranganathan, and Guevara Noubir. Security of GPS/INS based on-road location tracking systems. In *2019 IEEE Symposium on Security and Privacy (S&P)*, 2019.

[40] North American Van Lines, Inc. "Where Are Americans Moving in 2021?". https://www.northamerican.com/migration-map, 2022.

[41] Parmy Olson. Hacking A Phone's GPS May Have Just Got Easier. https://www.forbes.com/sites/parmyolson/2015/08/07/gps-spoofing-hackers-defcon/?sh=e73fe954efbf, 2015.

[42] Christina Pöpper, Nils Ole Tippenhauer, Boris Danev, and Srdjan Capkun. Investigation of signal and message manipulations on the wireless channel. In *Computer Security – ESORICS 2011*, 2011.

[43] Mark L. Psiaki, Steven P. Powell, and Brady W. O'Hanlon. GNSS spoofing detection using high-frequency antenna motion and carrier-phase data. In *proceedings of the 26th international technical meeting of the satellite division of the Institute of Navigation (ION GNSS+)*, 2013.

[44] Inc Qualcomm Technologies. Skyhook | Location Technology Provider. https://www.skyhook.com/, 2022.

[45] Aanjhan Ranganathan, Hildur Ólafsdóttir, and Srdjan Capkun. SPREE: A spoofing resistant GPS receiver. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016.

[46] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium (USENIX Security)*, Santa Clara, CA, 2019.

[47] Ignacio Sanchez, Riccardo Satta, Igor Nai Fovino, Gianmarco Baldini, Gary Steri, David Shaw, and Andrea Ciardulli. Privacy leakages in smart home wireless technologies. In *2014 International Carnahan Conference on Security Technology (ICCST)*, 2014.

[48] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security Symposium (USENIX Security)*, 2017.

[49] Bengt Sjölen and Gordan Savicic. "Packetbridge: Wireless geographical network intervention". https://criticalengineering.org/projects/packetbridge/, 2012.

[50] Navid Tadayon, Muhammed Tahsin Rahman, Shuo Han, Shahrokh Valaee, and Wei Yu. Decimeter ranging with channel state information. *IEEE Transactions on Wireless Communications*, 2019.

[51] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, 2011.

[52] Nils Ole Tippenhauer, Kasper Bonne Rasmussen, Christina Pöpper, and Srdjan Čapkun. Attacks on public WLAN-Based Positioning Systems. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2009.

[53] Mathy Vanhoef and Frank Piessens. Advanced Wi-Fi attacks using commodity hardware. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC)*. Association for Computing Machinery, 2014.

[54] Triet Dang Vo-Huu, Tien Dang Vo-Huu, and Guevara Noubir. Interleaving jamming in Wi-Fi networks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, 2016.

[55] John A. Volpe. Vulnerability assessment of the transportation infrastructure relying on Global Positioning System. https://rosap.ntl.bts.gov/view/dot/8435, 2001.

[56] Chen Wang, Chuyu Wang, Yingying Chen, Lei Xie, and Sanglu Lu. Smartphone privacy leakage of social relationships and demographics from surrounding access points. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[57] Kyle Wesson, Mark Rothlisberger, and Todd Humphreys. Practical cryptographic civil GPS signal authentication. *NAVIGATION, Journal of the Institute of Navigation*, 2012.

[58] Kyle D Wesson, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. An evaluation of the vestigial signal defense for civil GPS anti-spoofing. In *Proceedings of the 24th International Technical Meeting of the Satellite Division of The institute of navigation (ION GNSS)*, 2011.

[59] WiGLE.NET. "WiGLE: Wireless Network Mapping". https://wigle.net/index, 2022.

[60] Wikipedia contributors. Haversine formula — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=1075414115, 2022.

[61] Wikipedia contributors. Received signal strength indication — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Received_signal_strength_indication&oldid=1080897329, 2022.

[62] Bin Xu, Min Peng, Qing F. Zhou, and Xusheng Cheng. Fake access point localization based on optimal reference points. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 2018.

[63] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*. Association for Computing Machinery, 2005.

[64] Nian Xue, Liang Niu, Xianbin Hong, Zhen Li, Larissa Hoffaeller, and Christina Pöpper. Deepsim: Gps spoofing detection on uavs using satellite imagery matching. In *Annual Computer Security Applications Conference*, ACSAC '20, 2020.

[65] Kexiong (Curtis) Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. All your GPS are belong to us: Towards stealthy manipulation of road navigation systems. In *27th USENIX Security Symposium (USENIX Security)*, 2018.

[66] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 2013.