Research Article

Exploiting Proximity-Based Mobile Apps for Large-Scale Location Privacy Probing

Shuang Zhao,^{1,2} Xiapu Luo,³ Xiaobo Ma ,⁴ Bo Bai,^{1,2} Yankang Zhao,⁴ Wei Zou,^{1,2} Zeming Yang,^{1,2} Man Ho Au,³ and Xinliang Qiu⁵

¹Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

²School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

³Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong

⁴MOE KLINNS Lab, Xi'an Jiaotong University, Xi'an, China

⁵Beijing One Scorpion Cyber Security Co., Ltd., Beijing, China

Correspondence should be addressed to Xiaobo Ma; xma.cs@xjtu.edu.cn

Received 7 September 2017; Revised 17 December 2017; Accepted 27 December 2017; Published 14 February 2018

Academic Editor: Petros Nicopolitidis

Copyright © 2018 Shuang Zhao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Proximity-based apps have been changing the way people interact with each other in the physical world. To help people extend their social networks, proximity-based nearby-stranger (NS) apps that encourage people to make friends with nearby strangers have gained popularity recently. As another typical type of proximity-based apps, some ridesharing (RS) apps allowing drivers to search nearby passengers and get their ridesharing requests also become popular due to their contribution to economy and emission reduction. In this paper, we concentrate on the location privacy of proximity-based mobile apps. By analyzing the communication mechanism, we find that many apps of this type are vulnerable to large-scale location spoofing attack (LLSA). We accordingly propose three approaches to performing LLSA. To evaluate the threat of LLSA posed to proximity-based mobile apps, we perform real-world case studies against an NS app named Weibo and an RS app called Didi. The results show that our approaches can effectively and automatically collect a huge volume of users' locations or travel records, thereby demonstrating the severity of LLSA. We apply the LLSA approaches against nine popular proximity-based apps with millions of installations to evaluate the defense strength. We finally suggest possible countermeasures for the proposed attacks.

1. Introduction

As mobile devices with built-in positioning systems (e.g., GPS) are widely adopted, location-based mobile apps have been flourishing on the planet and easing our lives. In particular, recent years have witnessed the proliferation of a special category of such apps, namely, proximity-based apps, which offer various services by users' location proximity.

Proximity-based apps have gained their popularity in two (but not limited to) typical application scenarios with societal impact. One is location-based social network discovery, whereby users search and interact with strangers in their physical vicinity, and make social connections with the strangers. This application scenario is becoming increasingly popular, especially among the young [1]. Salient examples of mobile apps supporting this application scenario, which we

call *NS (nearby stranger) apps* for simplicity, include Wechat, Tinder, Badoo, MeetMe, Skout, Weibo, and Momo. The other is ridesharing (aka carpool) that aims to optimize the scheduling of real-time sharing of cars between drivers and passengers based on their location proximity. Ridesharing is a promising application since it not only boosts traffic efficiency and eases our lives but also has a great potential in mitigating air pollution due to its nature of sharing economy. Many mobile apps, such as Uber and Didi, are currently serving billions of people every day, and we call them *RS (ridesharing) apps* for simplicity.

Despite the popularity, these proximity-based apps are not without privacy leakage risks. For NS apps, when discovering nearby strangers, the user's exact location (e.g., GPS coordinates) will be uploaded to the app server and then exposed (usually obfuscated to coarse-grained relative

distances) to nearby strangers by the app server. While seeing nearby strangers, the user is meanwhile visible to these strangers, in the form of both limited user profiles and coarse-grained relative distances. At first glance, the users' exact locations would be secure as long as the app server is securely managed. However, there remains a risk of location privacy leakage when at least one of the following two potential threats happens. First, the location exposed to nearby strangers by the app server is not properly obfuscated. Second, the exact location can be deduced from (obfuscated) locations exposed to nearby strangers. For RS apps, a large number of travel requests consisting of user ID, departure time, departure place, and destination place from passengers are transmitted to the app server; then the app server will broadcast all these requests to drivers near users' departure places. If these travel requests were leaked to the adversary (e.g., a driver appearing everywhere) at scale, the user's privacy regarding route planning would be a big concern. An attacker can use the leaked privacy and location information to spy on others, which is our major concern.

In this paper, we systematically investigate the privacy leakage risks of typical proximity-based apps and perform case studies to prove that these privacy leakage risks can be exploited to spy on others. Note that the problem to be solved in this paper is not identifying location spoofing behavior in mobile apps but detecting location privacy leakage via location spoofing, since "spoofing" locations might be an official feature of an app, for example, booking Uber in advance for pickup from the airport or meeting/dating people in one's home town even though he/she is currently away. We find that existing proximity-based apps are vulnerable to large-scale location spoofing attack (LLSA) due to the insecure communication between the app and the server. Such insecure communication could be exploited by the adversary to perform automated and efficient location privacy probing at scale. We propose a series of methods to probe the location privacy of people using different proximity-based apps and show that our location probing methods are generally applicable to existing typical proximity-based apps. In addition to the insecure communication, we find that some apps surprisingly have careless design flaws harmful to privacy protection. We also perform case studies by performing attack testing against an NS app named Weibo and an RS app named Didi, for the purpose of demonstrating to what extent user privacy can be exposed and analyzed by the adversary. To help better prevent these privacy risks, we evaluate the defense strength of different proximity-based apps and suggest countermeasures to prevent the proposed attacks.

To the best of our knowledge, we are the first to conduct a systematic study of the location privacy leakage risk resulting from the insecure communication, as well as app design flaws, of existing typical proximity-based apps.

The major contributions include the following.

(i) *Track Location Information Flows and Evaluating the Risk of Location Privacy Leakage in Popular Proximity-Based Apps.* We analyze the location information flows from many aspects, including location accuracies, transport protocols, and packet contents, in popular NS apps such as Wechat,

Tinder, Skout, MeetMe, Momo, Mitalk, and Weibo and find that most of them have a high risk of location privacy leakage. Furthermore, we investigate an RS app named Didi, the largest ridesharing app that has taken over Uber China at \$35 billion dollars in 2016 and now serves more than 300 million unique passengers in 343 cities in China. We reveal that this app is also vulnerable to LLSA. The adversary, in the capacity of a driver, can collect a number of travel requests (i.e., user ID, departure time, departure place, and destination place) of nearby passengers. Our investigation indicates the broader existence of LLSA against proximity-based apps.

(ii) *Proposing Three General Attack Methods for Location Probing and Evaluating Them via Different Proximity-Based Apps.* We propose three general attack methods to probe and track users' location information, which can be applied to the majority of existing NS apps. We also discuss the scenarios for using different attack methods and demonstrate these methods on Wechat, Tinder, MeetMe, Weibo, and Mitalk separately. These attack methods are also generally applicable to Didi.

(iii) *Real-World Attack Testing against an NS App and an RS App.* Considering the privacy sensitivity of the user travel information, we present real-world attacks testing against Weibo and Didi so to collect a large amount of locations and ridesharing requests in Beijing, China. Furthermore, we perform in-depth analysis of the collected data to demonstrate that the adversary may derive insights that facilitate user privacy inference from the data.

(iv) *Defense Evaluation and Recommendation of Countermeasures.* We evaluate the practical defense strength against LLSA of popular apps under investigation. The results suggest that existing defense strength against LLSA is far from sufficient, making LLSA feasible and of low-cost for the adversary. Therefore, existing defense strength against LLSA needs to be further enhanced. We suggest countermeasures against these privacy leakage threats for proximity-based apps. In particular, from the perspective of the app operator who owns all users request data, we apply the anomaly-based method to detect LLSA against an NS app (i.e., Weibo). Despite its simplicity, the method is desired as a line-of-defense of LLSA and can raise the bar for performing LLSA.

Roadmap. Section 2 overviews proximity-based apps. Section 3 details three general attack approaches. Section 4 performs large-scale real-world attack testing against an NS app named Weibo. Section 5 shows that these attacks are also applicable to a popular RS app named Didi. We evaluate the defense strength of popular proximity-based apps and suggest countermeasures recommendations in Section 6. We present related work in Section 7 and conclude in Section 8.

2. Overview of Proximity-Based Apps

Nowadays, millions of people are using various location-based social network (LBSN) apps to share interesting location-embedded information with others in their social

networks, while simultaneously expanding their social networks with the new interdependency derived from their locations [1]. Most LBSN apps can be roughly divided into two categories (I and II). LBSN apps of category I (i.e., check-in apps) encourage users to share location-embedded information with their friends, such as Foursquare [2] and Google+ [3]. LBSN apps of category II (i.e., NS apps) concentrate on social network discovery. Such LBSN apps allow users to search and interact with strangers around based on their location proximity and make new friends. In this paper, we focus on LBSN apps of category II because they fit the characteristic of proximity-based apps.

For example, Wechat, which now has more than 540 million monthly active users around the world [4], has a feature called “Nearby.” This feature allows users to get a list of other users nearby as well as their coarse-grained relative distances. People can use this feature to discover strangers (and be discovered by others simultaneously) and then make friends with strangers of interest. Some apps (e.g., Facebook and Sina Weibo) that were not originally designed for NS are now also upgraded to this category. For example, Facebook Places was announced in 2010 to bring similar NS features into Facebook [5]. Sina Weibo, a Twitter-like microblog app in China, has also come up with a “Nearby” feature to let users discover nearby people, microblogs, and hot places.

In addition, most of the ridesharing apps such as Uber, Lyft, and Didi also use proximity information for nearby passenger or driver discovery; that is, the drivers can see nearby passengers, or the passengers can see nearby drivers. While sending a ridesharing request, the app will send the passenger’s geolocation to the server and the server will dispatch the request to nearby drivers based on the location proximity.

The workflow of social network discovery in NS apps is elaborated in Figure 1. The following steps will be performed in the scenario where a user searches for people nearby at location l_0 and time t_0 .

Step 1. The mobile app sends a request including the user’s current location l_0 which is obtained by GPS or online SDKs (e.g., Google SDK [6] and Baidu Location SDK [7]) and the authorization token to the server. The authorization token is provided by the server as a unique identifier as long as the user logs into the mobile app.

Step 2. Once the request from the user is received, the server saves the user’s location l_0 , time t_0 , and other information into the database for further usages, such as letting the user be visible to others.

Step 3. The server searches the database which contains the request time and locations of all the users who have ever searched for nearby people. Then, it finds out a list of users who are not in the friend list of the user ($user0$) and have appeared around location l_0 (within a distance of ΔD) less than a finite time ΔT ago. Given a user as u , the people in user u_i ’s social network as U_{fi} , and the distance between two locations l_i, l_j as D_{l_i, l_j} , the nearby users queried from the database for user u_0 can be described as follows:

$$\{u, l, t \mid u \notin U_{f0}, D_{l, l_0} \leq \Delta D, t \geq t_0 - \Delta T\}. \quad (1)$$

It should be noted that as long as a user has used the app to search nearby people, he/she can be found by other users around within a period of time, no matter the app is running in the foreground or background.

Step 4. The server sends a response to the mobile app with the queried results. For the purpose of privacy protection, the results returned by most NS app servers only contain essential user information u and coarse-grained distances l , because if the accurate distances are provided, a user’s exact location can be calculated by trilateration position methods easily [8]. Finally, the mobile app displays these results to the user.

Figure 2 shows the displayed results in typical NS apps: Wechat, Mitalk, Momo, Weibo, Skout, Tinder, Badoo, and LOVOO. The displayed user information normally contains nickname, gender, and other information (e.g., personalized signatures). In particular, Wechat, Mitalk, and Weibo provide distances to an accuracy of 100 meters and Momo does so to an accuracy of 10 meters, while Tinder provides distances accurate to within 0.1 miles. The user can view detailed information (e.g., publicly available photos) of nearby strangers, send greetings to them, and finally make new friends to extend the user’s own social network.

Figure 1 also presents two more scenarios to show in what circumstances a user can be found by others. In one scenario, where $user1$ searches for people nearby at a place close to the location of $user0$ (l_0), the searching time is short while ($\Delta t_1, t_1 < \Delta T$) after $user0$ searches for people nearby. According to (1), $user0$ can be found by $user1$ because $D_{l_0, l_1} \leq \Delta D$ and $t_0 \geq t_1 - \Delta T$. As for the other scenario where $user2$ searches for people nearby after a long while ($\Delta t_2, t_2 > \Delta T$), $user0$ cannot be found.

The workflow of passenger discovery in RS apps is similar to that of NS apps. Given a passenger, say p , the set of nearby passengers queried by the driver d_0 can be described by (2). Compared with (1), the major difference is that there is no U_{f0} in the case of ridesharing (due to lack of friend relationship), and ΔT depends on the time when the passenger’s ridesharing requests are processed, that is, accepted by a driver or canceled by the passenger. Another difference is that l would comprise the departure place and the destination place specified by a passenger:

$$\{p, l, t \mid D_{l, l_0} \leq \Delta D, t \geq t_0 - \Delta T\}. \quad (2)$$

However, such differences do not affect the way an attacker perform LLSA. Specifically, if an attacker u_A (whose friend list U_{fA} is empty) can send a fake location l_0 to the server in Step 1, he will get a response containing the passenger p around l_0 within the distance D_{l, l_0} in Step 4. By changing the value of l_0 constantly, the attacker can probe the passengers at any location.

In order to perform the location probing attack, we need to address the following challenging issues.

(i) *How to Forge the Request with Fake Locations.* We need to intercept the request in Step 1 and tamper the value of current location l . For securing data transportation, some NS apps

FIGURE 1: The workflow of social network discovery in proximity-based apps.

FIGURE 2: Search nearby people in NS apps.

use techniques like SSL authentication and data encryption, making request forgery a challenging task. Therefore, we need to try all possible ways to break or bypass these protection techniques.

(ii) *How to Perform a Large-Scale Probing Effectively and Economically.* We need to use as few resources as possible (e.g., 1 PC) to probe thousands of locations for large-scale attacks. Because the location information of the users will be cached for a while (ΔT) in Step 3, using too many probers at different locations synchronously is both resource-consuming and unnecessary. But if the time span of probing two nearby locations is too long (e.g., longer than ΔT), some data may be missed. For example, a user appeared at location l_0 at time t_0 , his location information can be probed only if the prober happens to probe at a location near l_0 between time t_0 and $t_0 + \Delta T$.

3. Location Privacy Probing via NS Apps

This section presents some general paradigms for location privacy probing via popular proximity-based NS apps. We first look deeply into some popular NS apps and examine the security through their transport protocols, request encryptions, response data, and so forth. Then, we propose and demonstrate three general methods for location privacy probing, which can be applied to the majority of existing NS apps.

3.1. Examining Popular NS Apps. We install nine popular NS apps including Badoo, LOVOO, MeetMe, Mitalk, Momo, Skout, Tinder, Wechat, and Weibo into Android/iOS mobile phones and use a web debugging proxy named Fiddler [9] to intercept and examine the network traffic between the apps and their servers. Table 1 shows the download counts of each app in Google Play (not available in China) and third party markets. These numbers indicate that Momo and Mitalk are popular in China, Badoo, Tinder, LOVOO, MeetMe, and Skout are popular in other countries, and Wechat and Weibo are popular in both China and other countries. We set up a proxy with Fiddler 4 on a computer and configure the proxy settings in the mobile phone to access Internet through our proxy. Then, all the HTTP/HTTPS traffic of the NS apps can be intercepted and monitored by Fiddler 4. Figure 3 shows the user interface of Fiddler 4. We see a list of intercepted HTTP/HTTPS requests on the left side of the user interface, including *Protocol*, *Host*, and *URL*. On the right side, there are two windows showing the details of the selected request and decoded response, respectively.

We examine the security of the intercepted network traffic from different aspects.

(i) *Transport Protocols.* The content in HTTP requests can be easily intercepted and manipulated to launch the request forgery attacks. HTTPS (HTTP over TLS/SSL) can provide data encryption to prevent the data from being tampered [10]. However, many apps do not correctly check the validity of the certificate. In this case, the HTTPS request can still be forged using a local self-signed certificate [11]. Some apps use SSL pinning to verify the certificate in order to prevent SSL

TABLE I: Examination results of popular NS apps.

APP	Originating place	Downloads (million) Google Play ¹	360 Android Market ²	Location accuracy in APP	Transport protocol	Request encryption	Location accuracy & other information in response data
Badoo	UK	100–500	0.81	None	Unknown	N/A	N/A
LOVOO	Germany	10–50	0.24	0.1 mi radius	HTTPS without SSL pinning	signature	100 m radius & last time
MeetMe	USA	10–50	0.08	100 m radius	HTTP with plaintexts	none	100 m radius
Mitalk	China	0.5–1	130	100 m radius	HTTP with plaintexts	checksum	10 m radius
Momo	China	1–5	1397	10 m radius	HTTPS without SSL pinning	none	1 m radius
Skout	USA	10–50	1.5	1000 m radius	HTTP with plaintexts	none	0.01 m radius
Tinder	USA	50–100	0.67	0.1 mi radius	HTTP with plaintexts	none	0.1 mi radius
Wechat	China	100–500	13463	100 m radius	Unknown	N/A	N/A
Weibo	China	10–50	4436	100 m radius	HTTP with plaintexts	none	0.00001° coordinate (≈ 1 m) & last time

¹ Most people in Chinese Mainland download Android apps from third-party markets because Google Play is inaccessible there; ² one of the largest Android third-party markets in China provided by Qihoo 360 Company (NYSE:QIHU).

FIGURE 3: Intercept and monitor network traffic with Fiddler 4.

man-in-the-middle attacks, but it can be bypassed using tools such as iOS-SSL-Kill-Switch [12] and Android-SSL-Trust-Killer [13]. Therefore, the content in HTTPS requests can also be intercepted and forged.

(ii) *Request Encryptions*. Another way for data protection is to encrypt some of the parameters in the HTTP or HTTPS request using proprietary algorithms. For example, in the HTTP request of Mitalk, there is a checksum parameter which is calculated using a proprietary algorithm. When some of the parameters in the request are tampered, it will be noticed by the server because the checksum value is erroneous. As long as the proprietary encryption algorithm is hard to crack (e.g., being compiled into .so file instead of .dex file), it can prevent the request from being tampered easily.

(iii) *Response Data*. The response data should not contain more information than what the app client needs. If the response data contains much more information (e.g., more accurate location than which is displayed in the app and the last time the person appeared), it will bring a risk of information leakage.

The analysis results are shown in Table 1. We can see that most apps use HTTP or HTTPS protocol without SSL pinning for data transportation and have no encrypted parameters in the requests. In this case, we can forge the HTTP/HTTPS requests to query nearby people at any location. Mitalk and LOVOO encrypt parameters (checksum and signature), and therefore the request can be forged only if we can crack the encryption algorithms and figure out the value of checksum or signature parameters. If the requests are too difficult to forge while the data protocol is unknown or the encryption algorithm is irreversible, we can also use mobile phone emulators and automated testing methods to simulate user actions to get people nearby at fake locations. The detailed demonstrations of these three methods are shown in Sections 3.2, 3.3, and 3.4.

3.2. Forging Requests. For NS apps, the request for searching people nearby contains parameters which are used to locate the user. The attacker can search people at any location by intercepting and tampering the location parameters. We demonstrate the attack in the following steps.

Step 1 (request interception). We use Fiddler as a web proxy to intercept the HTTP/HTTPS traffic between NS apps and

their servers. For HTTP traffic in plaintext, we can directly get the contents of the requests and responses. Fiddler can also decrypt HTTPS traffic, as long as a local self-signed certificate is generated and installed into the mobile phone. If certificate and public key pinning [14] is used in the NS app, reverse engineering work should be performed to replace the hard-coded key of the app with the one generated by Fiddler.

Some of the intercepted requests of different NS apps are as follows.

(i) *MeetMe*. GET <http://friends.meetme.com/mobile/boost/0?placement=meet&targetGender=b&latitude=38.988088&longitude=-76.977333&orderBy=distance&includeFriends=t&onlineOnly=f&pageSize=30>

(ii) *Weibo*. GET http://api.weibo.cn/2/place/nearby_users?gender=0&sourcetype=findfriend&offset=0&s=a5516ad4&c=android&lat=39.83178&long=116.290966&gsid=4u078d0a32pkzvo0r0ElvFLVM8j&page=1&sort=1&count=20

(iii) *Tinder*. Set current location:
POST <https://api.gotinder.com/user/ping>
"lat":39.73225467228202, "lon":116.1820556477647
Get nearby people:
GET <https://api.gotinder.com/recs/core>

(iv) *Momo*. POST <https://api.immomo.com>
Count=20&lat=39.83178&lng=116.290966&index=0

(v) *Skout*. Set current city name:
POST <http://and.skout.com/api/1/me/location>
Get nearby people:
GET http://and.skout.com/api/1/lookatme?application_code=3456025fd1e4ec43hec488b84fd700f4&area=city&limit=20&start=0&rand_token=3dcbf32a-9966-4b6b-9c18-441be07b12e1

In these requests, the location parameters {latitude, longitude} or {lat, long/lng/lon} indicate the location of the user who is searching nearby people.

Step 2 (request forgery). We forge HTTP or HTTPS requests by modifying the values of the location parameters in the intercepted requests to search nearby people at any location. We develop a program to automatically probe nearby people at random locations repeatedly. To avoid triggering the alarm of anomaly detection, the program sleeps for a short while after each probing.

Step 3 (response parsing). For most of the NS apps, the responses of searching nearby people are in JSON format because it is more efficient than XML and other data interchange formats [15]. We can extract useful information such as the person's id, name, distance, or geocoordinate by comparing the response data with the information displayed in the app.

Figure 4 shows the displayed results and the JSON-formatted response of searching nearby people in Weibo.

FIGURE 4: Displayed and JSON results of searching nearby people in Weibo.

FIGURE 5: Intercepted request and response of Mitalk.

Weibo provides distance values to an accuracy of 0.01 km. As shown in the figure, although we can see that the first user in the list is about 200 meters away, we cannot figure out the exact location of him only using this information. However, we find that the JSON-formatted response of Weibo exposes the geocoordinate of him as well as the time when he was located in that place for the last time (*last_at* field). It is indicated that the user's ID is 2753134315 and he was at the location (116.30042, 40.02080) at 01:09:58, Sep 27, 2015. Skout, Mitalk, and Momo have similar issues. That is, they provide more accurate distance values in the response data than in the apps.

3.3. Encryption Cracking. Some NS apps use data encryption techniques other than HTTPS protocol to secure the data traffic. They add encrypted parameters such as checksum or signature into the requests for data tampering detection. Take Mitalk, for example, the intercepted request of searching nearby people in Mitalk is shown in Figure 5, in which *latitude* and *longitude* represent the searching location. The JSON-formatted response contains an “ok” code and a list of persons around the searching location. However, when we try to modify the value of *latitude*, *longitude*, or any other parameter in the request, the response indicates errors with code 401.

After a series of experiments, we figure out that the parameter s in the request is generated by a customized algorithm and it represents the checksum of all other parameters. The server will recalculate the checksum and compare it to the

FIGURE 6: Inspect the layout hierarchy of Wechat with UIAutomator.

value of s when it receives a request. If the values do not match (i.e., one or more of the parameters might be tampered), an error message will be returned.

We decompile the APK of Mitalk into Java using tools including apktool [16], dex2jar [17], and Jd-gui [18] and perform reverse engineering to crack the algorithm of generating s . Then, we calculate the value of s to bypass the data tampering detection of the server and use the same method in Section 3.2 to search nearby people at any location.

3.4. Emulator Simulation. Some NS apps like Wechat and LOVOO use advanced encryption techniques which are difficult to crack. In this case, it will be too difficult, if not impossible, to intercept and forge the requests. Under these circumstances, we use mobile phone emulators and automated testing tools to simulate user's actions to probe nearby people at any location.

We demonstrate the method on Wechat using Android emulator [19] and *uiautomator*, which is a testing framework for Android [20]. We create an automated functional UI test case using *uiautomator*, which will automatically press a series of buttons to launch Wechat app and search nearby people in it. As soon as the results are displayed on the screen, the test case will inspect the UI to find the layout hierarchy and read information we need such as usernames and distances through the properties of specific UI components. The UI and the corresponding layout hierarchy of Wechat are shown in Figure 6. The algorithm of the test case is shown in Algorithm 1.

In our experiments, we first send fake geocoordinates to the emulator using a GPS command *geo fix* in the emulator’s control console and then launch the test case in the emulator to get nearby people at the fake location. By repeating the above two steps, we can probe nearby people at any location.

3.5. Location Tracking. As long as a large volume of data is collected, it is likely that a specific person would be probed multiple times at different places. Then, we can mark the location and the time when the person appeared on a map to track his/her locations.


```

(1) Press HOME button to return to home screen
(2) Find the icon with text "Wechat", click it and wait for new window
(3) Find the tab with text "Discover", click it and wait for new window
(4) Find the button with text "People Nearby", click it and wait for new window
(5) if Text "Unable to load your location data" is found then
(6)   Return error
(7) end if
(8) Get the listview L with resource id "com.tencent.mm:id/atf"
(9) Read the properties of each listitem in L to Result
(10) if L is scrollable then
(11)   Scroll down L for next page
(12)   Goto (9)
(13) else
(14)   Return Result
(15) end if

```

ALGORITHM 1: Search and read people nearby in Wechat.

For some NS apps such as Weibo, we can get the geo-coordinates of a targeted person directly. We mark the exact locations of the person with *points* on a map, as shown in Figure 7(a). For other apps like Wechat, Momo, and Tinder, we can only get coarse-grained locations which are determined by the probed location and the distances to the targeted person. In this case, we mark the approximate locations of the person with *circles*, as shown in Figure 7(b). The red points indicate the locations of the probers, and the circles denote the possible locations of the probed users. According to the trilateration positioning method [8], if a point lies on two circles at the same time, we can narrow down the possible locations to the intersections of the two circles. If a point lies on three or more circles, we can narrow down the possibilities to a unique point. Figure 7(b) also shows that, at nearly the same time, a user is probed by five probers (red points) and another user is probed by three probers. The locations of these two users can be deduced precisely to *Point1* and *Point2*.

4. Case Study: Real-World Attack Testing against an NS App Weibo

In this section, we demonstrate a large-scale real-world experiment of probing Weibo users all over the most area of Beijing, China. In Weibo app, a user's extract geocoordinate will be exposed when using the "Nearby" function to search nearby people or tweets. As is shown in Figure 8, we generate 896 probing points (the red dots in the figure) inside 5th Ring Road of Beijing covering about 870 km² and run a program with one PC to walk through these probing points one by one randomly and search nearby people. At last we have probed nearly 50 million data including *id*, *nickname*, *coordinate*, and *lasttime* of more than 400 thousand unique users.

The time distribution of the probed data is shown in Figure 9(a). The higher values of the time distribution occur during 18:00 to 24:00, in which the peak values appear at around 23:00, while the lower values occur during 1:00 to 7:00. It also reflects the regularity of people's activities; that

is, people have more social interactions with others from dusk (after work) till midnight (before sleep) than in the daytime. Figure 9(b) shows the heatmaps of probed people in different places at different hours, from dark to light as blue-green-yellow-red. The first subpicture at the top-left shows the density of probed people in Beijing during 0:00–0:59, and so on. The lighter color indicates the higher density of probed people. From Figure 9(b) we can get a similar conclusion as the one from Figure 9(a), that is, there are more people who use the "Nearby" feature from dusk to midnight than those in the daytime. Besides, we can also see that there are more people who are far away from the downtown than those in downtown from midnight to morning (0:00–9:00), and more people in downtown at day and evening, because the business areas including companies and malls are mainly concentrated in downtown and most of the residential areas are built far from the center of the city.

We carried out an experiment with 10 volunteers whose location have been probed, in which we compare the probed locations with their real-life frequent places. As is shown in Figure 10, 33% of the probed locations are around their workplaces, while 42% are near their dwellings. It indicates that, for Weibo, people are more likely to search nearby people or tweets at home or workplaces.

In order to recognize the location patterns of the probed people, we use DBSCAN (Density-Based Spatial Clustering of Applications with Noise), which is a density-based clustering algorithm [21], to cluster the different locations which are close to each other (e.g., less than 1 km) into one location area. Figure 11(a) shows an example of 10 locations being clustered into 3 areas, and Figure 11(b) shows the statistical result of location clusters of the probed people. Statistics suggest that 76.5% of the people can only be found in one consistent area, 15.8% of the people can be found in two different areas, and 7.7% of the people can be probed in 3 or more areas.

For the people who are often found in one consistent area, we can deduce their privacy information by the probed time and locations following the assumptions:

FIGURE 7: Location tracking via different NS apps.

FIGURE 8: Probing Points of Weibo in Beijing.

(1) If the probed person is often found in the consistent area at daytime, it is likely that the person works near the probed locations.

(2) If the probed person is often found in the consistent area at night, it is likely that the person lives near the probed locations.

For the people who are often found in two consistent areas, if the probed person is often found in one area at daytime and, in the other at night, the home/work location pair can be deduced, which can be used for reidentification of the user [22].

At the end of the experiment, we analyze the probed locations of some verified celebrity accounts for two reasons. Firstly, the location privacy is extremely important to the celebrity because they are not willing to let the public know

where they live and where they have been unless they explicitly release relevant information, and the exposure of their location privacy may affect social order. Besides, we can evaluate the accuracy of location privacy probing by comparing the probed locations to related information from the Internet (e.g., the address of the company, the location of a celebrity event).

Some of the celebrity accounts whose locations are probed are shown in Table 2. We can see that the actress, doctor, and TV host are probed many times in many places. It indicates that they often search nearby people or places (in Weibo, the function of searching nearby people and searching nearby places coexists) in different areas (e.g., finding hot restaurants when arriving at a new place). Meanwhile, other accounts use “nearby” functions much less frequently. Take

(a) Time Distribution of probed data in Beijing

(b) Heatmap of probed people in Beijing during 0:00 to 23:00

FIGURE 9: Distribution of probed data and people.

FIGURE 10: Probing locations of volunteers.

TABLE 2: Some celebrity Weibo accounts whose location are probed.

ID	Verified type	Number of fans (followers)	Probed times	Probed locations
2411*****4	Company	9906619	2	2
1223*****0	Singer	3719265	1	1
1231*****4	Actress	4038654	21	14
1327*****3	CEO	3221332	11	11
1054*****7	Doctor	718481	14	14
1874*****2	Sportsman	614322	2	2
1558*****4	TV Host	1073046	28	19

FIGURE 11: Location clustering.

FIGURE 12: Probed locations of “360 mobile assistant.”

the account “2411* * * * 4” as an example; it is the verified official account of 360 mobile assistant, which belongs to Qihoo 360 company. The probed locations of the account are marked with red points in Figure 12, while the company’s actual address is marked with a star. We find out the probed locations are essentially the same as the related public information, which proves the efficiency of our work.

5. Location Privacy Probing via RS Apps

Besides NS apps, some ridesharing apps may also be vulnerable to LLSA. In ridesharing apps, there are two communication styles for the drivers to receive orders of nearby passengers.

Push-Style. The server pushes a ridesharing request to one or more drivers close to the passenger. In this scenario, the

driver can get only one ridesharing request at a time and choose to accept it or not. Uber and Lyft belong to this type.

Pull-Style. The driver pulls ridesharing requests from the passengers close to him/her from the server. In this scenario, the driver can get a list of many ridesharing requests at a time, and choose one or none of them to accept as needed. Didi’s ridesharing service falls into this category.

For both communication styles above, one can fabricate fake locations to get ridesharing requests at any place. However, for push-style ridesharing apps, it would be ineffective for the attacker to perform LLSA, because the driver can only get *one* (rather than all) ridesharing request at a time at any place. In contrast, for pull-style ridesharing apps, the driver can only get all ridesharing requests at a time at any place. Therefore, these apps would be good vantage points for the attacker to perform LLSA.

5.1. Data Probing. The methods of data probing via RS apps are similar to those of NS apps.

5.1.1. Uber. Uber is one of the most popular ridesharing apps all over the world. While getting ridesharing requests, Uber will firstly get the user’s current location via a map SDK. Specifically, in China, Uber will send a POST request to `http://loc.map.baidu.com/sdk.php`, and get a JSON-formatted response containing the geolocation. After that, Uber will communicate with its server using SSL to get new ridesharing requests. So we can intercept and modify the response from `http://loc.map.baidu.com/sdk.php` to fake the user’s location and use the emulator simulation method similar to what is described in Section 3.4 to probe ridesharing requests at different places.

5.1.2. Didi. Didi is one of the most popular ride-hailing apps in China, which has taken over Uber’s business in China since October, 2016. It provides different kinds of functions including taxi-hailing, limousine service (similar to Uber Black), private-car service (similar to Uber X) and ridesharing service. We focus on the ridesharing service because it uses a pull-style communication mechanism, which is much makes it easier for us to perform LLSA.

In the ridesharing service, the driver shares a ride with passengers, and the passengers will split the cost with the driver in return. Thus, it is much cheaper than others. Moreover, it is supported by the government because it can substantially reduce traffic pressure in peak hours, while, for limousine services and private-car services, the drivers are required to have business operation licenses to avoid huge financial penalties.

In Didi, there are two kinds of ridesharing orders, namely, the following.

(1) *Along-the-Route Orders.* The route of the passenger is similar to one of the driver’s regular routes; for example, both the departure place and the destination place are near the driver’s preset ones.

FIGURE 13: Searching nearby ridesharing passengers in Didi.

(2) *Nearby Orders*. Only the departure place is near the driver. When searching nearby passengers, the driver can get a list of ridesharing requests, each of which consists of the passenger's nickname, departure place, destination place, departure time, and price, as is shown in Figure 13(a). Furthermore, while viewing the detailed information of the ridesharing request, the driver can get the detailed departure geocoordinate and destination geocoordinate on a map, as is shown in Figure 13(b). The driver then can select to accept one of the requests. Once a ridesharing request is accepted, the driver can have the phone number of the passenger and contact with him/her.

When a driver is searching nearby passengers, the app will send an HTTP request with the driver's geocoordinate to the server and will receive a JSON-formatted response containing *order_id*, *passenger_id*, *from_lng*, *from_lat*, *to_lng*, *to_lat*, *setup_time*, and so forth. *passenger_id* is the unique ID of the passenger. *from_lng* and *from_lat* represent the departure place of the passenger. *to_lng* and *to_lat* indicate the destination place of the passenger. *setup_time* represents the time when the passenger is about to set off.

The intercepted HTTP request of searching nearby ridesharing passengers in Didi is as follows:

```
GET http://api.didialift.com/beatles/api/
driver/order/nearbylist?appversion=4.3.4
&datatype=101&filter=5&lat=39.731833&lng
=116.187432&locatePerm=1&num=100&offset_
order_id=0&token=...
```

So, we can use request forgery method similar to what is described in Section 3.2 to perform LLSA via Didi.

5.2. Case Study: Real-World Attack Testing against Didi. In this section, considering the privacy sensitivity of the user travel information, we perform a real-world attack testing

against Didi so as to collect travel requests in Beijing and perform in-depth analysis of the collected data to demonstrate that the adversary may derive insights from the data that facilitates user privacy inference.

We do not choose Uber to perform the case study for the reasons as follows.

(i) As described above, Uber is a push-style ridesharing app, so the attacker has to register many Uber accounts for large-scale LLSA. We do not have many driver licenses to register Uber's driver accounts.

(ii) Uber has a strict means of cheating detection and an extremely heavy penalty for cheating. Specifically, Uber will ban the drivers account forever, while Didi will ban the account only for a while, upon the successful detection of cheating. So the cost of the LLSA via Uber is quite high.

We generate 3190 probing points all over Beijing including the downtown and county districts, covering about 8,370 km², as is shown in Figure 14. The distance between neighbored probe points is 2 km. Then we run a program on a PC to send forged HTTP requests to get nearby ridesharing requests on these probing points and get 763,370 requests from 423,067 unique passengers.

The time distribution of the probed ridesharing requests in one week is shown in Figure 15. At weekends, the ridesharing requests are quite evenly spread between 9:00 and 22:00, and there are no evident peaks. On workdays, there are often two peaks. One is 7:00–8:00 in the morning, and the other is 17:00–18:00 in the afternoon. They are coincident with the rush hours in Beijing. It is to be observed that on July 20, the number of ridesharing requests between 15:00 and 20:00 is extremely high in comparison with that in other days. That is because there was a big rainstorm from 9:00 to 20:00 in Beijing, and the transportation system was partially paralyzed due to road waterlog problems in the afternoon. So many people go off work earlier than normal.

FIGURE 14: The probed area of Didi ridesharing service in Beijing.

FIGURE 15: The time distribution of probed ridesharing requests.

(a) Ridesharing routines in the morning (b) Ridesharing routines in the evening

FIGURE 16: Probed ridesharing routines in different time of day.

We draw some animated pictures to show the routines of the ridesharing requests in different times of day. Figure 16(a) shows the requests between 7:00 and 8:00 in the morning, where most of the routine directions are from suburbs to downtown, while Figure 16(b) shows the requests between 19:00 and 20:00 in the evening, where most of the routine directions are from downtown to suburbs. These also reflect that business areas are mainly concentrated in downtown and most residential areas are far from the center of the city.

To look into the area distribution of the ridesharing requests, we collect the departure geocoordinates and destination geocoordinates of all probed requests and draw the

routines on a map. Figure 17(a) shows the routines of all ridesharing requests during a single day. The lighter the area, the more the requests. Most of the destination or departure places are in the urban area. Besides, there are some request-intensive places in the suburbs (marked with circles), which also represent population intensive residence areas. These facts reflect the population distribution to a certain extent: the northern suburbs have a larger population than the southern suburbs. Figure 17(b) shows the CDF of the ridesharing distances. It is illustrated that more than 90% of the ridesharing distances are less than 40 km, and nearly 70% of them are more than 10 km.

FIGURE 17: Distribution of probed ridesharing requests.

FIGURE 18: Ridesharing requests in business-intensive areas.

Specifically, we choose three typical business-intensive areas in the center, north and south of Beijing separately to further study the patterns of routines and the distances. The results are shown in Figure 18. For Zhongguancun, Haidian District, which is in the center of Beijing, the ridesharing requests are nearly from/to all directions. What interests us is that the ridesharing requests from/to Chuangxin Road, Changping District, are mostly to/from the south or east, while Chuangxin Road is in the northwest of Beijing. Coincidentally, the ridesharing requests from/to Yizhuang BDA, Daxing District, are mostly to/from the north or west, while Yizhuang BDA is in the southeast of Beijing. It indicates that

many people need to go across downtown for work, which is one of the reasons for traffic jam in peak hours.

The distance distributions of the ridesharing requests probed in each areas are shown in the bar figures under the maps in Figure 18, respectively. They show that the ridesharing distances are mostly between 10 km and 50 km.

As long as we get enough ridesharing requests, we can track a passenger according to his/her id or nickname by analyzing all his/her ridesharing routines on a map.

Figure 19 shows the probed ridesharing requests during 10 days of a unique person whose ID is 269***81. The departure and destination places of all his ridesharing requests

FIGURE 19: Ridesharing requests of a unique person.

concentrate in three places: $(40.076 \pm 0.001, 116.418 \pm 0.002)$, $(39.743 \pm 0.002, 116.182 \pm 0.001)$, and $(39.983, 116.317)$, which are marked as *Place A*, *Place B*, and *Place C* separately. It is indicated from the figure that the person often travels from *Place A* to *Place B* in the morning and leaves *Place B* for *Place A* in the evening. Therefore, we can deduce that the person lives in *Place A* and works in *Place B*. By searching these two places on a map, further information can be found that *Place A* is near Tiantongyuan, which is one of the largest residential areas in Beijing, and *Place B* is Beijing Institute of Technology. As long as we get that the person may be a teacher who works in Beijing Institute of Technology and lives in Tiantongyuan, it is much easier to locate his real identity.

We can further explore the correlation among the passengers' travel behaviors based on clustering analysis. Such correlation could reveal passengers' diurnal group activities that are likely to be related to their social activities. Diurnal group activities can be considered as a set of travel records with similar departure times and close departure locations (i.e., diurnal departure group, DPG) or with similar arrival times and close destination locations (i.e., diurnal destination group, DSG), during a day. In this context, departure/arrival times are in the form of hour:time:second, exclusive of the date, thereby allowing a departure/destination group to contain records of different days. Intuitively, passengers in a departure group with departure times in going-to-work (going-off-work, resp.) hours may live nearby (work nearby, resp.), while passengers in a destination group with arrival times in going-to-work (going-off-work, resp.) hours may work nearby (live nearby, resp.).

To derive DPG and DSG, we need to cluster passengers' records of (departure time, departure longitude, and departure latitude) and (arrival time, destination longitude, and destination latitude), respectively. In both cases, we denote the records to cluster by $(t, \text{lng}, \text{lat})$. Here, the challenge lies in that t is in time domain, but lng and lat are in space domain. In order to cluster records close to each other in both time and space domain, we define the distance function of two records r_i and r_j as follows:

$$d(r_i, r_j) = \sqrt{[\alpha(t_i - t_j)]^2 + [M_{\text{lng}}(\text{lng}_i - \text{lng}_j)]^2 + [M_{\text{lat}}(\text{lat}_i - \text{lat}_j)]^2}, \quad (3)$$

where M_{lng} and M_{lat} are constant values denoting the distance (i.e., km) per unit of longitude and latitude, respectively, and α is a tunable (positive) parameter that equivalently transforms t into the space domain. A large value of α would increase the sensitivity of $d(r_i, r_j)$ to the time difference of two records, that is, $(t_i - t_j)$, thereby encouraging records with small time differences to be grouped into one cluster. Particularly in the case of $\alpha = 0$, records with departure/destination places close to each other would be grouped into one cluster, regardless of their time differences.

We then leverage the DBSCAN algorithm, which finds core samples of high density and expands clusters from them, to cluster passengers' records collected from July 19 to Aug 1, 2016. There are totally 460,881 unique user IDs and 850,826 records, among which 619,634 records are collected on weekdays and 231,192 records on weekends. The two key parameters of DBSCAN, namely, *min_pts* and *eps*, jointly define a core sample as a sample whose distance is smaller than *eps* to at least *min_pts* samples. In our experiment, we empirically set *min_pts* as 10, *eps* as 0.1 km, and α as 10 km/h. Such a parameter setting generates clusters where records in each cluster are reasonably close, due to small values of mean (pairwise) location distance and mean (pairwise) time difference. As illustrated in Figure 20, a node represents a cluster (DPG or DSG). We observe that the mean location distance of DPGs ranges from 0 to 0.2 km, and the mean time difference of DPGs ranges from 0 to 0.09 hour (i.e., 5.4 minutes). In the context of DSGs, the range of the former is narrowed (i.e., from 0 to 0.12 km), while the range of the latter is enlarged (i.e., from 0 to 0.8 hours). However, the mean time difference of most DSGs centers around 0.05 hours (i.e., 3 minutes) and does not exceed 0.2 hours (i.e., 12 minutes), thereby having small values in the vast majority of cases.

Figure 21 presents the clustering results regarding DPG and DSG on weekdays and weekends. The x -axis is the cluster labels and the y -axis is the user ID space. A node means the user belongs to the cluster on the corresponding axis. The color of a node represents the number of records that the user generates in the cluster, where darker colors indicate more records. The clusters on the x -axis are in ascending order of their sizes from left to right. Figures 21(a) and 21(c) show that the clustering results on weekdays, where 12,928

FIGURE 20: The mean (pairwise) location distance versus the mean (pairwise) time difference of records in each cluster when $min_pts = 10$, $eps = 0.1$, and $\alpha = 10$. Each node represents a cluster (DPG or DSG). The small values of mean location distance and mean time difference for each cluster demonstrate that the records in each cluster are reasonably close.

FIGURE 21: The clustering results regarding DPG and DSG at weekdays and weekends. The x -axis is the cluster labels and the y -axis is the user IDs. A node corresponds to a user's travel request, and its color represents the number of records that a user generates in a cluster, where darker colors means more records.

users are grouped into 703 clusters and 15,320 users are grouped into 690 clusters, respectively. In Figures 21(a) and 21(c), we observe that a user may generate travel requests belonging to many different clusters (i.e., different DPGs) on weekdays. This indicates that the diversity of users' departure places and times. In Figures 21(b) and 21(d), we observe reduced numbers of clusters and users belonging to a cluster. This indicates that users' departure/arrival places and times are less coordinated on weekends. This is because people during weekends have more freedom to schedule their traveling activities. We present an interesting example observed from the largest cluster in Figure 21(b). In this cluster, more than 100 passengers depart from a famous view place on Sunday around 14:00 to different places. This probably indicates that these passengers finish their travels and go back home. The results demonstrated in these figures reveal that social outbreaks which may indicate public events or public emergencies could be observed by collecting a large number of individual location records, potentially facilitating better security surveillance by security forces.

In Didi, for the purpose of privacy protection, the passenger's phone number is invisible in the list of nearby ridesharing requests. However, the driver who accepts the ridesharing request and successfully gets the order can get the passenger's phone number in order to get contact with him/her. Therefore, given a ID or nickname of a targeted person, we can search all his/her ongoing ridesharing requests all over the city. Once we find his/her ongoing request, we can automatically accept the request and get the ridesharing order. Then we can naturally get the phone number of the targeted person, based on which we can find out his/her real name (phone numbers are required to be registered with real names in China). After that, we will cancel the ridesharing order because we actually do not have a real car to pick up him/her.

However, if the driver accepts and then cancels ridesharing requests too many times per day, the account of the driver may be banned temporarily or permanently. So it is still difficult to get the phone number of many passengers at the same time using one driver account.

6. Defense Evaluation and Recommendations on Countermeasures

In this section, we evaluate the overall risk induced by proximity-based apps as well as the defense strength of some of them. Then, we discuss some possible countermeasures against the threat of location privacy leakage via proximity-based apps.

6.1. Risk Evaluation. We evaluate the risk of exploitation and data leakage induced by the investigated proximity-based apps. If an app can be exploited by forging requests without reverse engineering, it has a high risk of exploitation. Meanwhile, it is difficult to perform large-scale location spoofing attack with emulator simulation because the simulation is time-consuming in forging requests. It is also difficult for normal attackers to crack the encryption algorithm of an app. So, if an app can only be exploited by encryption cracking or emulator simulation, it has a medium risk of exploitation

because only sophisticated attackers or attackers who have a lot of computers and app accounts to run emulators can do that. As for data leakage, if an app will leak people's geocoordinate or location with high accuracy (e.g., within 10 m) in LLSA, it has a high risk of data leakage because the attacker can get people's precise locations directly. If the leaked location is coarse-grained, the risk of data leakage is medium because the attacker needs to probe more data and use trilateration positioning to get people's relatively precise locations.

As shown in Table 3, more than 1/2 of the apps (i.e., 6 out of 11) have a high risk of exploitation. Meanwhile, more than 1/3 of the apps (i.e., 4 out of 11) can expose people's location privacy with high accuracy, hence having a high risk of data leakage.

6.2. Defense Evaluation. In our large-scale probing experiments with Mitalk, Momo, Wechat, Weibo, and Didi which have much more users in Beijing than other apps, it is observed that, for all these apps, probing without intermission would trigger anomaly detection. Mitalk, Momo, and Weibo will ban the abnormal account for a short while, that is, several minutes, while Didi will ban the request IP for several hours. Wechat has a much more strict penalty, that is, lock the account until the user relogin and unlock it manually. In order to avoid the abnormal behavior penalties, the probing speed must be reduced. After several experiments, we get the approximate safe probing rate for these apps, as is shown in Table 4.

For Mitalk and Weibo, we set an interval of 2-3 seconds between each probing request to successfully avoid the anomaly detection. For Didi, the prober's IP will be banned only if the prober keeps sending forged requests without intermission for several hours. So we let the prober sleep 0-2 seconds randomly every time it makes 5 places probed. This also successfully evades the detection. However, Momo only allows non-VIP accounts to send about 1000 "search nearby" requests per day, which increase the cost of data probing dramatically. Lastly, the probe speed for Wechat is much slower because we have to use the emulator simulation method to get nearby users, and the speed under this circumstance is slower, that is, 60 seconds per place. Even so, the account used for probing is locked after several hours.

6.3. Recommendations on Countermeasures. First, using HTTP protocol with plaintexts for data transportation is extremely unsafe, because it is vulnerable to both request forgery and MITM (man-in-the-middle) attacks. Besides, although HTTPS protocol can provide data encryption during transmission, misusing TLS/SSL in developing apps such as allowing all hostnames, trusting all certificates, SSL stripping, and lazy SSL usage [23] will make the apps fail to verify the certificate and be thus vulnerable to TLS MITM attacks. Using encrypted parameters or proprietary protocol which is difficult to crack can increase the attack cost dramatically.

Second, antiprobing and anomaly detection methods should be used by the service providers to distinguish automatic probers from normal human users. It is not efficient

TABLE 3: Risk evaluation of proximity-based apps.

APP	Possible exploit method	Risk of exploitation	Risk of data leakage
Badoo	Emulator simulation	Medium	Medium
LOVOO	Encryption cracking & forging requests, or emulator simulation	Medium	Medium
MeetMe	Forging requests	High	Medium
Mitalk	Encryption cracking & forging requests, or emulator simulation	Medium	High
Momo	Forging requests	High	High
Skout	Forging requests	High	Medium
Tinder	Forging requests	High	Medium
Wechat	Emulator simulation	Medium	Medium
Weibo	Forging requests	High	High
Didi	Forging requests	High	High
Uber	Emulator simulation	Medium	Medium

TABLE 4: Defense evaluation of proximity-based apps.

APP	Exploit method	Penalty for abnormal behaviors	Approximate safe probing speed
Mitalk	Encryption cracking & forging requests	Ban the account for several minutes	3 seconds/place
Momo	Forging requests	Ban the account for several minutes	1 second/place Allow around 1000 requests per day for non-VIP users
Wechat	Emulator simulation	Lock the account. Need to re-login and unlock it manually	Slower than 60 seconds/place
Weibo	Forging requests	Ban the account for several minutes	2 seconds/place
Didi	Forging requests	Ban the IP for several hours	0.5 seconds/place

enough to simply limit the quota for searching nearby people of each user just as what Momo does, because it can be bypassed easily by using multiple probing accounts and devices. A witty designed machine behavior model should be studied and applied for better detection and protection [24–26]. For example, at to the NS app, one may randomize some items of his/her profiles (e.g., nickname, user ID) to disconnect the linkage between user IDs and locations each time when he/she looks for nearby strangers [27], hence preventing an attacker from inferring the privacy even if the location is leaked.

From the perspective of the app operator who owns all users request data, the abnormal users that may be an attacker conducting LLSA need to be detected. To this end, the app operator may want to distinguish normal user profiles from abnormal ones and then train a classifier to predict whether a user is abnormal or not. We next leverage the anomaly-based detection method to demonstrate the feasibility of identifying the attacker who conducts LLSA.

In order to verify the feasibility of such anomaly detection, we collect location data of the NS app Weibo users who look for nearby strangers and meanwhile generate synthetic data simulating the behavior of an attacker conducting LLSA. The Weibo location data, consisting of 59,793,831 locations records of 526,533 unique users in the city area of a large metropolis Beijing, was collected from March 9, 2015, for 90 days. Each record is a 4-tuple consisting of time, user ID, user nickname, and GPS coordinates. We consider the collected Weibo data generated by normal users, as is based upon the assumption that there is no attacker conducting LLSA against Weibo (since we are the first to report the vulnerability of Weibo to LLSA as far as we know).

On the other hand, the synthetic data simulating the behavior of an attacker is generated using the following heuristic strategies. First, the attacker conducts LLSA with random intervals t uniformly distributed between 0 and T , and we vary the values of T (e.g., from 10 seconds to 86,400 seconds with a step of 10 seconds) to simulate different attackers. Second, the fake location that each time the attacker uses

can be either randomly selected in Beijing, or selected within the ϵ -neighborhood (e.g., $\epsilon = tV$) of the last fake location, where V is the user's moving speed, which we randomly choose from 0 to $V_{\max}$ (e.g., 60 km/h). Using these strategies, we generate the synthetic attack data. Then, we build a two-class classifier using SVM to distinguish the attackers from normal users, with features covering maximum time interval, minimum time interval, average time interval, maximum distance, and minimum distance between two consecutive location exposures, as well as total location exposure counts. Our extensive experiments using tenfold cross-validation results in surprisingly good detection performance (i.e., detection rate around 99% and false positive rate less than 0.5%). These results show the feasibility and the promising of the anomaly detection method, which could further raise the bar for performing LLSA. As a future work, we will consider more sophisticated attack strategies that may be stealthy so to evade the detection, while being simultaneously efficient in probing location privacy.

Besides the countermeasures above from the perspective of securing communication and anomaly detection, countermeasures dedicated to validating the correction of locations are desired. One possible solution is to validate the locations based on multiple (opportunistic) information sources. For example, besides the GPS coordinates, the app is meanwhile required to provide other location-dependent information, such as the MAC address of the WiFi or the parameters of the 3G/LET base station that it connects. In this way, the app server can validate the locations based on the consistency of these information from multiple sources.

Last but not least, in the client/server (C/S) model, when responding to the request, the response data volume should be kept minimal without more extra information than needed by the apps.

7. Related Work

There is a rich literature of privacy protection in proximity-based social networks wherein users interact with each other

based on their physical distances. For example, Gruteser and Grunwald [28] proposed a quadtree-based anonymity algorithm that can decrease the spatial resolution of users' locations. Duckham and Kulik [29] used obfuscation to achieve a balance between the utility of proximity-based services and users' privacy. Mascetti et al. [30] proposed a spatial generalization algorithm named *grid* to protect users' privacy in location-based services. Xu and Cai [31] proposed a feeling-based position privacy protection method by adding dummy requests with faked indistinguishable locations. Ghinita et al. [32] pointed out some drawbacks of an anonymity method for privacy protection in location-based services and present a novel method for location-dependent queries based on Private Information Retrieval (PIR), which can provide protection against correlation attacks. All these studies cannot fully address the privacy threat posed by LLSA, because their focus is obfuscating the locations rather than securing the communication to prevent LLSA.

Dong et al. [33] developed a secure protocol for social network discovery in proximity-based networks. The proximity computation protocol they developed can preserve the privacy of social coordinates and social proximity, while simultaneously providing coordinate verification and efficient filtering. In their design, users cannot forge social coordinates and any user can authenticate another user's identity and social coordinate. Although such a design is sophisticated enough to prevent LLSA, our investigation shows that existing proximity-based apps have not yet incorporated the design.

A few studies focus on how to utilize information in proximity-based networks for social engineering and person reidentification. Li and Chen [34] performed comprehensive analysis over user profiles, social graphs, and attribute correlations using proximity-based social network traces collected from a company. Jedrzejczyk et al. [35] argued that the location data in proximity-based social networks collected anonymously would lead to significant security vulnerabilities. People using NS apps can be reidentified by cross-referencing their location data with related information available. Li et al. [36] developed an automated system *FreeTrack* to track users' locations via proximity-based NS apps such as Wechat, Skout, and Momo. They carry out proof-of-concept attacks by employing Android virtual machines to fabricate fake locations to get the coarse-grained distances of target persons and then calculate the precise location using iterative trilateration. Leveraging Android virtual machines to fabricate fake locations are not so efficient as directly sending bulk packets carrying fake locations to the app server by exploiting the insecure communication. Our work facilitates these studies in that it provides efficient approaches to collecting large-scale user location information needed by these studies. Note studies such as [37–39] also focus on user location data analysis. However, they crawl check-in data (explicitly published by users) from social networks like Foursquare and Twitter, instead of using location data from proximity-based apps.

Several recent studies have been attracted by the privacy assessment and protection in proximity-based ridesharing. For example, Friginal et al. [40] designed dynamic carpooling

with enhanced privacy protection. Aïvodji [41] claimed that it is not safe that the ridesharing data is stored in a database managed by the app operators. They introduced a privacy-preserving ridesharing system to help solving the problem. Our work differs from these studies because our focus is how to breach the privacy of a ridesharing app by exploiting the communication between the app and the server without hacking the server's database. By extending LLSA to a ridesharing app, we show that the privacy risks can exist even if the database is securely managed. Although in [40] Friginal et al. indeed pointed out that attackers can infer the location of a user's home and workplace by tracking the user's movements in ridesharing apps, they did not go a step further due to the lack of actual ridesharing data, while our actual data from real-world attack testing against Didi confirms their point.

8. Conclusion

In this paper, we investigated the privacy leakage risks of proximity-based apps, including the apps with functionalities of searching nearby strangers and ridesharing. We examined popular proximity-based apps and found that they could be exploited for launching large-scale location spoofing attack due to the insecure communication between the app and the server. We proposed three general methods for conducting such attacks via proximity-based apps. Moreover, we also found that some apps may contain fine-grained sensitive information more than needed in the raw data returned by the server to the app, thereby further increasing the privacy leakage risks. Using the proposed attack methods, we evaluated the overall risk induced by popular proximity-based apps and derived insightful observations beneficial to privacy protection of existing proximity-based apps. Our evaluation showed that current privacy protection in proximity-based apps are insufficient. We discussed and proposed possible protection mechanisms to address the privacy risks.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported in part by Key Laboratory of Network Assessment Technology, CAS, and Beijing Key Laboratory of Network Security and Protection Technology. This work is supported in part by National Key Research and Development Program (2016YFB0801004 and 2016QY07405), National Natural Science Foundation (61602371, 61221063, and 61202396), China Postdoctoral Science Foundation (2015M582663), Natural Science Basic Research Plan in Shaanxi Province (2016JQ6034), the Fundamental Research Funds for the Central Universities, Shaanxi Province Postdoctoral Science Foundation, the Hong Kong GRF (no. PolyU 152279/16E), and the HKPolyU Research Grants (G-YBJX) of China.

References

- [1] Y. Zheng, "Tutorial on location-based social networks," in *Proceedings of the 21st international conference on World wide web*, vol. 12, 2012.
- [2] Foursquare Inc, <https://foursquare.com/about>.
- [3] M. Hattersley, *Google+ Companion*, John Wiley & Sons, 2012.
- [4] Statista Inc, Number of active wechat messenger accounts 2010–2015, <http://www.statista.com/statistics/255778/number-of-active-wechat-messenger-accounts/>.
- [5] J. O'Dell, *A Field Guide to Using Facebook Places*, Aug 2012, <http://mashable.com/2010/08/18/facebook-places-guide/#hXTFxQjU78qq>.
- [6] R. Rogers, J. Lombardo, Z. Mednieks, and B. Meike, *Android Application Development: Programming with the Google SDK*, O'Reilly Media, Inc., 2009.
- [7] Baidu Inc, Baidu location sdk, <http://api.map.baidu.com/lbsapi/cloud/geosdk.htm>.
- [8] W. Murphy and W. Hereman, *Determination of a Position in Three Dimensions Using Trilateration and Approximate Distances*, Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, Colorado, 1995.
- [9] E. Lawrence, *Fiddler: Web Debugging Proxy*, 2007.
- [10] K. Hickman and T. Elgamal, *The ssl protocol*, vol. 501, Netscape Communications Corp, 1995.
- [11] N. Rudrappa, Defeating ssl certificate validation for android applications.
- [12] Github, ios-ssl-kill-switch, <https://github.com/iSECPartners/ios-ssl-kill-switch>.
- [13] Github, Android-ssl-trust-killer ssl kill switch, <https://github.com/iSECPartners/Android-SSL-TrustKiller>.
- [14] C. Evans, C. Palmer, and R. Sleevi, "Public Key Pinning Extension for HTTP" RFC Editor RFC7469, 2015.
- [15] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML data interchange formats: A case study," in *Proceedings of the 22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009*, pp. 157–162, USA, November 2009.
- [16] R. Winsniewski, *Android-apktool: A tool for reverse engineering android apk files*, 2012.
- [17] B. Alll and C. Tumbleson, *Dex2jar: Tools to work with android. dex and java. class files*.
- [18] E. Dupuy, Jd-gui: Yet another fast java decompiler, 2012, <http://java.decompiler.free.fr/?q=jdgui/>.
- [19] Android Developers. Using the android emulator, 2012.
- [20] Android Developers. UiAutomator, 2013.
- [21] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "Density-based spatial clustering of applications with noise," in *Proceedings of the Int. Conf. Knowledge Discovery and Data Mining*, vol. 240, 1996.
- [22] P. Golle and K. Partridge, "On the anonymity of home/work location Pairs," in *Proceedings of the 7th International Conference on Pervasive Computing*, pp. 390–397, Berlin, Germany, 2009.
- [23] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why Eve and Mallory love Android: An analysis of Android SSL (in)security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012*, pp. 50–61, USA, October 2012.
- [24] Q. Li and G. Cao, "Providing privacy-aware incentives in mobile sensing systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 6, pp. 1485–1498, 2016.
- [25] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao, "Defending against sybil devices in crowdsourced mapping services," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '16*, pp. 179–191, New York, NY, USA, June 2016.
- [26] K. Fawaz and K. G. Shin, "Location privacy protection for smart-phone users," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS '14)*, pp. 239–250, USA, November 2014.
- [27] T. Jeske, "Floating car data from smartphones: What google and waze know about you and how hackers can control traffic," in *Blackhat*, 2013.
- [28] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, pp. 31–42, ACM, San Francisco, Calif, USA, May 2003.
- [29] M. Duckham and L. Kulik, "A formal model of obfuscation and negotiation for location privacy," in *Proceedings of International Conference of Pervasive Computing (LNCS '05)*, pp. 152–170, Munich, Germany, May 2005.
- [30] S. Mascetti, C. Bettini, D. Freni, and X. S. Wang, "Spatial generalisation algorithms for LBS privacy preservation," *Journal of Location Based Services*, vol. 1, no. 3, pp. 179–207, 2007.
- [31] T. Xu and Y. Cai, "Feeling-based location privacy protection for location-based services," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 348–357, ACM, Chicago, Ill, USA, November 2009.
- [32] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pp. 121–132, ACM, 2008.
- [33] W. Dong, V. Dave, L. Qiu, and Y. Zhang, "Secure friend discovery in mobile social networks," in *Proceedings of the IEEE INFOCOM*, pp. 1647–1655, April 2011.
- [34] N. Li and G. Chen, "Analysis of a location-based social network," in *Proceedings of the 2009 IEEE International Conference on Social Computing, SocialCom 2009*, pp. 263–270, Canada, August 2009.
- [35] L. Jedrzejczyk, B. A. Price, A. K. Bandara, and B. Nuseibeh, "On the impact of real-time feedback on users' behaviour in mobile location-sharing applications," in *Proceedings of the the Sixth Symposium*, p. 1, Redmond, Washington, July 2010.
- [36] M. Li, H. Zhu, Z. Gao et al., "All your Location are Belong to Us: Breaking mobile social networks for automated user location tracking," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2014*, pp. 43–52, USA, August 2014.
- [37] B. Carbunar, R. Sion, R. Potharaju, and M. Ehsan, "Private badges for geosocial networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 10, pp. 2382–2396, 2014.
- [38] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1082–1090, ACM, August 2011.
- [39] C. Zhiyuan, J. Caverlee, L. Kyumin, and D. Z. Sui, "Exploring millions of footprints in location sharing services," *ICWSM*, vol. 2011, pp. 81–88, 2011.

- [40] J. Friginal, S. Gambs, J. Guiochet, and M.-O. Killijian, "Towards privacy-driven design of a dynamic carpooling system," *Pervasive and Mobile Computing*, vol. 14, pp. 71–82, 2014.
- [41] U. M. Aïvodji, "Privacy enhancing technologies for ridesharing," in *Proceedings of the Student Forum of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.