I Know Where You All Are! Exploiting Mobile Social Apps for Large-Scale Location Privacy Probing

Shuang Zhao[1,5], Xiapu Luo[3,4], Bo Bai[1,5], Xiaobo Ma[2,3,4(✉)], Wei Zou[1,5], Xinliang Qiu[1,5], and Man Ho Au[3,4]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{zhaoshuang,baibo,zouwei,qiuxinliang}@iie.ac.cn
[2] MOE KLINNS Lab, Xi'an Jiaotong University, Xi'an, China
[3] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
xma.cs@xjtu.edu.cn
[4] The Hong Kong Polytechnic University Shenzhen Research Institute, Shenzhen, China
[5] Beijing Key Laboratory of Network Security and Protection Technology, Beijing, China
{csxluo,csallen}@comp.polyu.edu.hk

Abstract. Mobile social apps have been changing the way people interact with each other in the physical world. To help people extend their social networks, Location-Based Social Network (LBSN) apps (e.g., Wechat, SayHi, Momo) that encourage people to make friends with nearby strangers have gained their popularity recently. They provide a "Nearby" feature for a user to find other users near him/her. While seeing other users, the user, as well as his/her coarse-grained relative location, will also be visible in the "Nearby" feature of other users. Leveraging this observation, in this paper, we model the location probing attacks, and propose three approaches to perform large-scale such attacks on LBSN apps. Moreover, we apply the new approaches in the risk assessment of eight popular LBSN apps, each of which has millions of installation. The results demonstrate the severity of such attacks. More precisely, our approaches can collect a huge volume of users' location information effectively and automatically, which can be exploited to invade users' privacy. This study sheds light on the research of protecting users' private location information.

1 Introduction

With the wide adoption of mobile devices with built-in positioning systems (e.g., GPS), LBSNs (Location-based Social Networks) are becoming increasingly popular, especially among the young. Nowadays, millions of people are using various 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].

J.K. Liu and R. Steinfeld (Eds.): ACISP 2016, Part I, LNCS 9722, pp. 3–19, 2016.
DOI: 10.1007/978-3-319-40253-6_1

These LBSN apps can be roughly divided into two categories (I and II). LBSN apps of category I encourage users to share location-embedded information with their friends, such as Foursquare and Google+. Foursquare, which has achieved more than 55 million users worldwide since 2009 [2], allows users to *check-in* at some interesting places and then share the check-in locations with their friends. Google+, besides sharing check-in locations, even allows users to share their real-time locations with pre-specified users (e.g., their families) [3].

LBSN apps of category II concentrate on location-based social network discovery. Such LBSN apps allow users to search and interact with strangers around, and make new friends. Salient examples of this category include Wechat, Momo, SayHi, Skout and so forth. 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 locations. 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 LBSNs are now also upgraded to this category. For example, Facebook Places was announced in 2010 to bring similar LSBN 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.

While using LBSN apps of category I to check-in or share locations with friends, users are likely to *explicitly* publish their locations to their social networks [6]. On the contrary, while using LBSN apps of category II to discover nearby users, users will get information *without explicitly* making their locations public. As a matter of fact, when a user (using LBSN apps of category II) searches nearby users, the user's exact location (e.g., GPS coordinates) will be uploaded to the app server, and then exposed (usually after obfuscation as needed) to nearby users by the app server. 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 users by the app server is not properly obfuscated. Second, the exact location can be deduced from (obfuscated) locations exposed to nearby users.

In this paper, we systematically investigate these two threats using typical LBSN apps of category II. We find that existing LBSN apps[1] are vulnerable to these threats, which could be exploited by the adversary to perform automated and efficient large-scale location probing attacks, Such attacks could reveal the location of any user that uses the "Nearby" feature. We propose a series of novel methods to probe the location privacy of people using different LBSN apps and show that our location probing methods are general and applicable to the vast majority of existing LBSN apps.

[1] Unless otherwise mentioned, LBSN apps investigated in the remainder of this paper belong to category II by default.

Our work is different from existing work in twofold. First, we are able to probe locations of any user whereas in existing works such as [7–9] the attacker can only probe the locations of his/her friends. Second, we propose three general approaches for location probing, and discuss the scenarios for using different approaches, whereas existing studies usually focus on specific situations. For example, [10] uses Android virtual machines to carry out proof-of-concept attacks via Wechat, Skout and Momo, and [11] discusses the possibilities of launching the sniffing attacks if the HTTP traffic of LBSN apps can be intercepted and manipulated.

To the best of our knowledge, we are the first to carry out a large-scale experiment to evaluate the practical efficiency of such attack in real world. Our contributions include:

- **Track location information flows and evaluate the risk of location privacy leakage in popular LBSN apps.** We analyze the location information flows from many aspects including location accuracies, transport protocols, packet contents, etc. in popular LBSN apps such as Wechat, Momo, Mitalk, SayHi, Skout, MeetMe and Weibo, and find out that most of them have high risk of location privacy leakage.
- **Propose three general attack methods for location probing and evaluate them via different LBSN apps.** We propose three general attack methods to probe and track users' location information, which can be applied to the majority of existing LBSN apps. We also discuss the scenarios for using different attack methods, and demonstrate these methods on SayHi, Mitalk and Wechat, separately.
- **Recommendation of count-measures.** We suggest some count-measures against this new threat of privacy leakage in LBSNs.

The rest of the paper proceeds as follows. Section 2 presents an overview of location-based social networks and LBSN apps. Section 3 details three general attack approaches with examples. After recommending some countermeasures in Sect. 4, we conclude the paper in Sect. 5.

2 Overview of LBSN Apps

Most LBSN apps have two features: check-in and social network discovery. We focus on the latter because people's check-in locations are shared with their friends and therefore they are not regarded as private information.

The workflow of social network discovery in LBSN apps is elaborated in Fig. 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 (a):** 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 [12], Baidu Location SDK [13]) and the authorization token to the server. The authorization token is provided by the server as a unique identifier as long as the user logins into the mobile app.

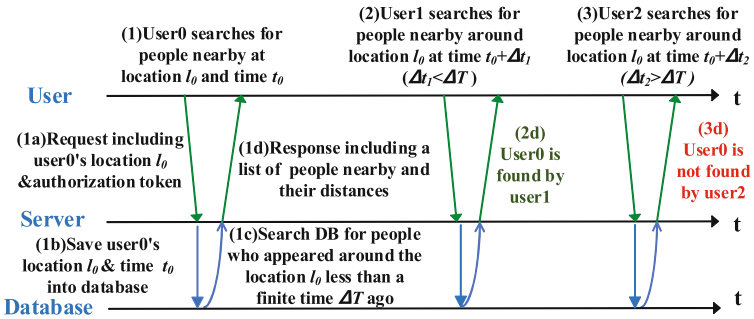


Fig. 1. The workflow of social network discovery in LBSN apps

- **Step (b):** 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 use, such as letting the user be visible to others.
- **Step (c):** 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 the 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:

$$\{u, l, t | u \notin U_{f0}, D_{l,l_0} \leqslant \Delta D, t \geqslant t_0 - \Delta T\} \tag{1}$$

- **Step (d):** The server sends a response to the mobile app with the queried results. For the purpose of privacy protection, the results returned by most LBSN 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 [14]. Finally, the mobile app displays these results to the user.

Figure 2 shows the displayed results in typical LBSN apps: Wechat, Mitalk, Momo, Weibo, SKout, SayHi, Badoo and LOVOO. The displayed user information generally 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 and SayHi do so to an accuracy of 10 meters. However, LOVOO provides distances accurate to within 0.1 miles, which is the least accurate.

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 other 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 which is close to the location of *user0* (l_0) for a short while

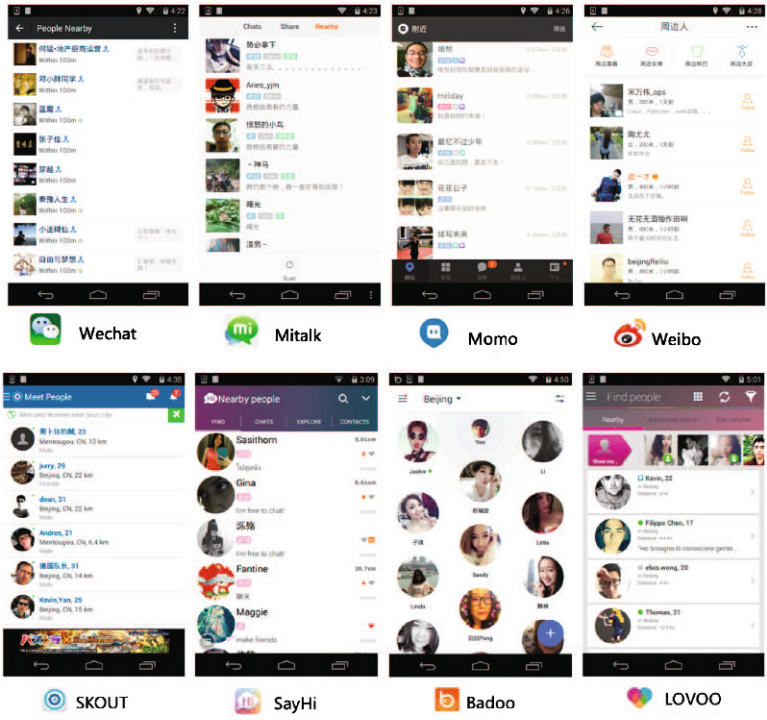


Fig. 2. Search nearby people in LBSN apps

($\Delta t_1, t_1 < \Delta T$) after *user0* searches for people nearby, according to Eq. 1, *user0* can be found by *user1* because $D_{l_0,l_1} \leqslant \Delta D$ and $t_0 \geqslant t_1 - \Delta T$. 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.

If an attacker u_A (whose friend list U_{fA} is empty) is able to send a fake location l_A to the server in Step (a), he will get a response containing the users u_i around l_A and their distances D_{l_i,l_A} in Step (d). By changing the value of l_A constantly, the attacker can probe the users at any location.

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

- **How to forge the request with fake locations:** We need to intercept the request in Step (a), and tamper the value of current location l. For securing data transportation, some LBSN 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.
- **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 (c), using too many probers to probe 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 LBSN Apps

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

3.1 Examining Popular LBSN Apps

We install 8 popular LBSN apps including LOVOO, MeetMe, Mitalk, Momo, SayHi, Skout, Wechat and Weibo into an Android mobile phone, and use a web debugging proxy named Fiddler [15] to intercept and examine the network traffic between the apps and their servers. We set up a proxy with Fiddler 4 on a computer, and configure the proxy settings in the Android mobile phone to access Internet through our proxy. Then, all the HTTP/HTTPS traffic of the LBSN 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*, *URL*, etc. On the right side, there are two windows showing the details of the selected request and decoded response respectively.

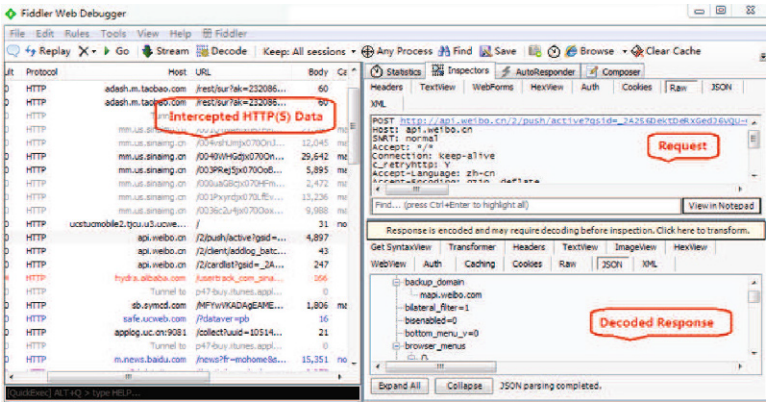


Fig. 3. Intercept and monitor network traffic with Fiddler 4

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

- **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 [16]. It is worth noting that TLS/SSL can be configured either one-way or two-way. In one-way TLS/SSL communication, the server is required to present the certificate to the client, but the client is not required to present the certificate to the server, meaning that the server will accept the request from anyone. In this case, the HTTPS request can still be forged using a local self-signed certificate [17]. In two-way SSL communication, both the server and the client are required to present the certificates, which makes request interception a hard task. Therefore, we consider the HTTP and HTTPS with one-way SSL protocols are insecure, and HTTPS with two-way SSL protocol is safe.
- **Request encryptions:** Another way for data protection is to encrypt some of the parameters in the HTTP request. For example, a checksum or signature can prevent the request from being tampered effectively, as long as the encryption algorithm can not be cracked easily.
- **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, 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 with one-way SSL 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 is transported via HTTPS with two-way SSL 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 Sects. 3.2, 3.3 and 3.4.

3.2 Forging Requests

For LBSN 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 LBSN apps and their servers. For HTTP traffic in

Table 1. Examination results of popular LBSN apps

APP	Downloads(million)			Location Accuracy in APP	Transport protocol	Request encryption	Location accuracy &Other information in response data
	Google play[1]	App Store	360 Android market[2]				
LOVOO	14	8.4	0.001	0.1 mi radius	HTTPS with one-way SSL	signature	100m radius&last time
MeetMe	14	8.9	0.001	100 m radius	HTTP with plaintexts	none	100 m radius
Mitalk	0.8	1.8	17	100 m radius	HTTP with plaintexts	checksum	10 m radius
Momo	1.8	26	168	10 m radius	HTTPS with one-way SSL	none	1 m radius
SayHi	12	3.3	0.04	10 m radius	HTTP with plaintexts	none	0.000001° coordinate ($\approx$ 0.1m)
Skout	23	24	0.06	1000 m radius	HTTP with plaintexts	none	0.01 m radius
Wechat	169	43	455	100 m radius	HTTPS with two-way SSL	N/A	N/A
Weibo	10	23	456	100 m radius	HTTP with plaintexts	none	0.00001° coordinate ($\approx$ 1m) &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.

plaintext, we can directly get the contents of the requests and responses. Fiddler can also decrypt HTTPS traffic with one-way SSL, as long as a local self-signed certificate is generated and installed into the mobile phone. If certificate and public key pinning [18] is used in the LBSN 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 LBSN apps are as follows:

- **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
- **SayHi:**
  GET http://r.x-vv.com/ft?s=L99uLTNuWxp1RJ&gt=true&ii=1&ts=0&of=0&lc=116.2085999,39.9726842
- **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=u078d0a32pkzvoOr0ElvfLVM8j&&page=1&sort=1&count=20
- **Momo:**
  POST https://api.immomo.com
  `Count=20&lat=39.83178&lng=116.290966&index=0`
- **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

- **Badoo:**
  POST https://eu1.badoo.com/jsss/mjinba.phtml?v=2
  `<jsondata>`
  In these requests, the location parameters {*latitude,longitude*}, *lc*, {*lat,long*}, {*lat,lng*} indicate the location of the user who is searching nearby people.

Step 2: Request Forgery. We forge HTTP or HTTPS with one-way SSL 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. In order to avoid the alarm of anomaly detections, the program sleeps for a short while after each probing.

Step3: Response Parsing. For most of the LBSN apps, the responses of searching nearby people are in JSON format because it is more efficient than XML and other data interchange formats [19]. We can extract useful information such as the person's id, name, distance or geo-coordinate 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 SayHi. SayHi provides distance values to an accuracy of 0.01 km. As shown in Fig. 4, although we can see that the user *Sasithorn* is 5.01 km away, we cannot figure out the exact location of *Sasithorn* only using this information. However, we find that the geo-coordinate of *Sasithorn* (116.339193, 39.9923481) is in the JSON-formatted response data.

Besides the geo-coordinate of the user, the JSON-formatted response of Weibo also exposes the time when the user was located in that place for the last time (*last_at* field). Figure 5 demonstrates a real-world example, which indicates that the user with ID *2753134315* was at the location (116.30042, 40.02080) at 01:09:58, Sep 27th, 2015.

3.3 Encryption Cracking

Some LBSN 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 Fig. 6, 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.

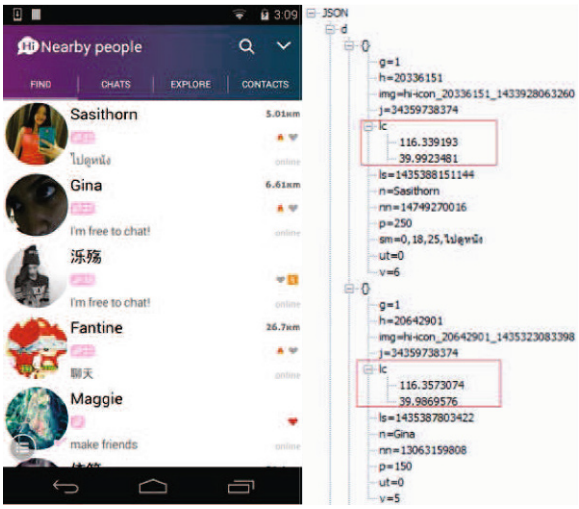


Fig. 4. Displayed and JSON results of searching nearby people in SayHi

Fig. 5. Displayed and JSON results of searching nearby people in Weibo

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

We perform reverse engineering to crack the algorithm of generating parameter s. We decompile the APK of Mitalk into Java using tools including apktool [20], dex2jar [21] and Jd-gui [22], and find out the generation procedure of s, which is shown in Fig. 7(a).

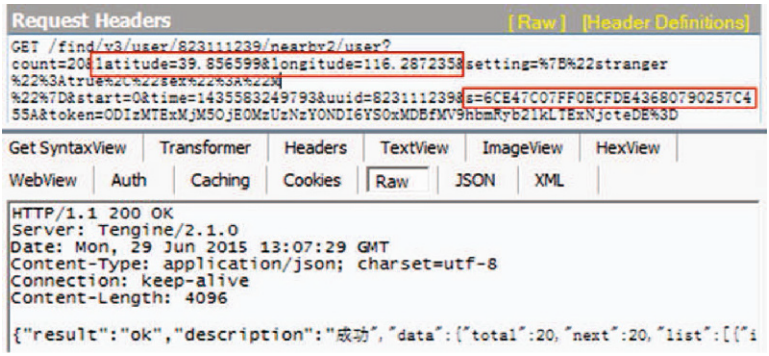
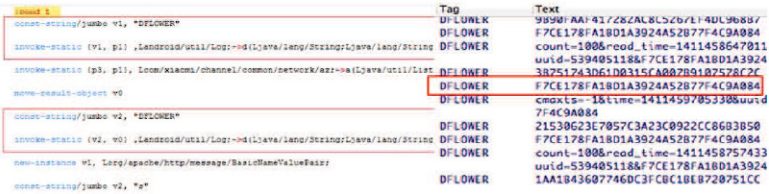


Fig. 6. Intercepted request and response of Mitalk

```
public static String a(List<NameValuePair> paramList, String paramString)
{
  paramList.add(new BasicNameValuePair("time", String.valueOf(System.currentTimeMillis())));
  Collections.sort(paramList, new bb());
  StringBuilder localStringBuilder = new StringBuilder();
  Iterator localIterator = paramList.iterator();
  for (int m = 1; localIterator.hasNext(); m = 0)
  {
    NameValuePair localNameValuePair = (NameValuePair)localIterator.next();
    if (m == 0) {
      localStringBuilder.append("&");
    }
    localStringBuilder.append(localNameValuePair.getName()).append("=").append(localNameValuePair.getValue());
  }
  localStringBuilder.append("&").append(paramString);
  return com.xiaomi.channel.d.f.c(new String[com.xiaomi.channel.d.f.a.2]localStringBuilder.toString().getBytes()));
}
```

(a) The Decompiled Generation Procedure of Parameter *s*



(b) Monitor *paramString* in Logcat

Fig. 7. Cracking encrypted parameter *s* of Mitalk

From the figure we conclude that the encrypted parameter *s* is calculated according to Eq. 2:

$$s = b(name1 = value1\&...\&nameN = valueN\&paramString) \tag{2}$$

In the equation, *name1* to *nameN* are the alphabetical parameters of the request excluding *s*. Meanwhile, *paramString* is a fixed value. In order to get the value of *paramString*, we firstly disassemble the APK file of Mitalk into smali codes, and insert some debug codes to let the app print the value of *paramString* into *logcat* (which is an Android logging system for collecting debug outputs) while runing. Then, we repackage [23] the APK file and install it into an Android phone. When

```
public static char[] b(byte[] var0, int var1, int var2)
    {
        int var3 = (3 * var2 * 4) / 3;
        char[] var4 = new char[4 * ((var2 + 2) / 3)];
        int var5 = var1 + var2;

        int var10;
        for (int var6 = 0; var1 < var5; var1 = var10)
        {
            int var7 = var1 + 1;
            int var8 = 255 & var0[var1];
            int var9;
            if (var7 < var5)
            {
                int var21 = var7 + 1;
                var9 = 255 & var0[var7];
                var7 = var21;
            }
            else
            {
                var9 = 0;
            }

            int var11;
            if (var7 < var5)
            {
                var10 = var7 + 1;
                var11 = 255 & var0[var7];
            }
            else
            {
                var10 = var7;
                var11 = 0;
            }

            int var12 = var8 / 4;
            int var13 = (var8 & 3) << 4 | var9 / 16;
            int var14 = (var9 & 15) << 2 | var11 / 64;
            int var15 = var11 & 63;
            int var16 = var6 + 1;
            var4[var6] = b[var12];
            int var17 = var16 + 1;
            var4[var16] = b[var13];
            char var18;
            if (var17 < var3)
            {
                var18 = b[var14];
            }
            else
            {
                var18 = (char)61;
            }

            var4[var17] = var18;
            int var19 = var17 + 1;
            char var20;
            if (var19 < var3)
            {
                var20 = b[var15];
            }
            else
            {
                var20 = (char)61;
            }

            var4[var19] = var20;
            var6 = var19 + 1;
        }

        return var4;
    }
```

Fig. 8. Decompiled function *b* from *com.xiaomi.channel.d.f.a.b*

we launch the repackaged Mitalk app and login with an account, the value of *paramString* can be watched in a logcat viewer, as shown in Fig. 7(b).

From *com.xiaomi.channel.d.f.a.b*, the function *b* in Eq. 2 can be decompiled. The decompiled function is shown in Fig. 8. We can calculate the value of *s* using Eq. (2) to bypass the data tampering detection of the server, and then use the same method in Sect. 3.2 to search nearby people at any location.

3.4 Emulator Simulation

Some LBSN apps like Wechat and LOVOO use HTTPS with two-way SSL protocol or use advanced encryption techniques. 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 [24] and *uiautomator*, which is a testing framework for Android [25]. 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 Fig. 9. The algorithm of the testcase is shown in Algorithm 1.

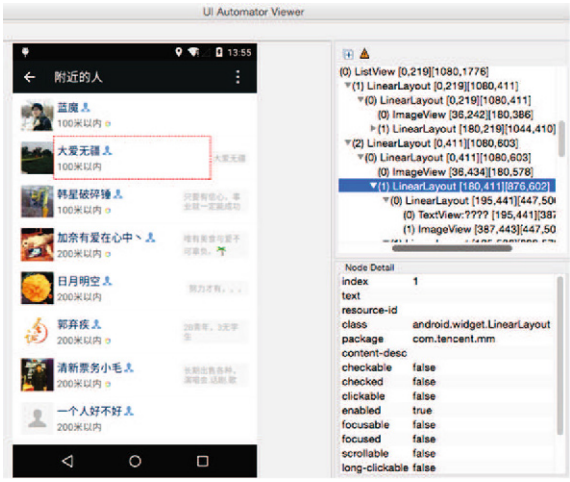


Fig. 9. Inspect the layout hierarchy of Wechat with UIAutomator

Algorithm 1. Searching and Reading People Nearby in Wechat

```
Press HOME button to return to home screen
Find the icon with text "Wechat", click it and wait for new window
Find the tab with text "Discover", click it and wait for new window
Find the button with text "People Nearby", click it and wait for new window
if Text "Unable to load your location data" is found then
    Return error
end if
Get the listview L with resource id "com.tencent.mm:id/atf"
Read the properties of each listitem in L to Result
if L is scrollable then
    Scroll down L for next page
    Goto 1
else
    Return Result
end if
```

In our experiments, we first send fake geo-coordinates to the emulator using a GPS command *geo fix* in the emulator's control console, and then launch the testcase 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

(a) Weibo (b) Wechat (c) Momo

Fig. 10. Location tracking via different LBSN apps

location and the time when the person appeared on a map to track his/her locations.

For some LBSN apps such as Weibo and SayHi, we can get the geo-coordinates of a targeted person directly, and hence we mark the exact locations of the person with *points* on a map, as shown in Fig. 10(a). For some other apps such as Wechat and Momo, we can only get the coarse-grained locations which are determined by the probed location and the distances from the targeted person to the probed location. In this case, we mark the approximate locations of the person with *circles* on a map, as shown in Fig. 10(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 [14], 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 10(c) shows that at nearly the same time, a user is probed by 5 probers (red points) and another user is probed by 3 probers. The locations of these two users can be deduced precisely to *Point1* and *Point2*.

3.6 Risk Evaluation

We next evaluate the overall risk induced by the popular LBSN apps. Table 2 shows the overall risk evaluation of popular LBSN apps. More than half of the apps (i.e., five out of eight) are easy to exploit. Meanwhile, half of the apps (i.e., four out of eight) can expose people's location privacy with high accuracy. Weibo and SayHi have the highest risks, because they can be exploited for location probing easily and meanwhile can leak people's geo-coordinate directly. LOVOO and Wechat have a relatively low risk of being exploited for large-scale location probing, because the efficiency of emulator simulation is much lower than forging requests, and they only expose people's coarse-grained locations.

Table 2. Risk evaluation of popular LBSN apps

APP	Possible exploit method	Exploit difficulty	Accuracy of leaked data
LOVOO	Encryption cracking & Forging requests or emulator simulation	Difficult	Medium
MeetMe	Forging requests	Easy	Medium
Mitalk	Encryption cracking & Forging requests or emulator simulation	Difficult	High
Momo	Forging requests	Easy	High
SayHi	Forging requests	Easy	Very high
Skout	Forging requests	Easy	Low
Wechat	Emulator simulation	Difficult	Medium
Weibo	Forging requests	Easy	Very high

4 Recommendations on Counter-Measures

In this section, we discuss some possible counter-measures against the threat of location privacy leakage via LBSN apps.

Firstly, we point out that 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, since the one-way TLS/SSL authorization does not require the server to check the validity of the certificate from the client, a self-signed local certificate can be generated and used to parse the plain content from the TLS/SSL traffic, which makes HTTPS protocol with one-way TLS/SSL unsafe to use. Other misuses of TLS/SSL in the development of the apps such as allowing all hostnames, trusting all certificates, SSL stripping and lazy SSL usage [26] will also make the apps vulnerable.

Also, anti-probing and anomaly detection methods should be used by the service providers to distinguish automatic probers from normal human users. It is not efficient 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. Last but not least, in the client/server (C/S) model, while responding to the request, the response data volume should be small without more extra information than the app client needs, while leaving the data filtering and omitting work to the client will bring the risk of information leakage.

5 Conclusion

In this paper, we pointed out that mobile social apps will introduce location privacy leakage when they provide the functionality of searching nearby people.

We examined the risks of location leakage in popular LBSN apps and find out that they can be exploited for launching the location probing attacks. Moreover, we proposed three general methods for conducting such attacks via LBSN apps.

Using the new attack methods, we evaluated the overall risk induced by popular LBSN apps and had many interesting observations. This study shows that the current methods for location privacy protection in LBSN apps are insufficient and new protection mechanisms are desired to address such risk.

Acknowledgements. This work is supported in part by the Hong Kong GRF/ECS (No. PolyU 5389/13E) and the HKPolyU Research Grant (G-YBJX), and in part by the National Natural Science Foundation (No. 61202396, 61221063, U1301254), Postdoctoral Science Foundation (2015M582663), Postdoctoral Science Foundation of Shaanxi Province, 863 High Tech Development Plan (2012AA011003), 111 International Collaboration Program, the Fundamental Research Funds for the Central Universities (3115200112), Shenzhen City Science and Technology R&D Fund (JCYJ20150630115257892), and Technology Innovation Project of Chinese Academy of Sciences (Y5X0011716, Y5X0011516), of China.

References

1. Zheng, Y.: Tutorial on location-based social networks. In: Proceedings of the 21st International Conference on World Wide Web, WWW, vol. 12 (2012)
2. Foursquare Inc. About us. https://foursquare.com/about
3. Hattersley, M.: Google+ Companion. Wiley, Indianapolis (2012)
4. Inc, S.: Number of active wechat messenger accounts 2010–2015. http://www.statista.com/statistics/255778/number-of-active-wechat-messenger-accounts/
5. O'Dell, J.: A field guide to using facebook places. Mashable. com, p. 23 (2012)
6. Zhao, X., Li, L., Xue, G.: Checking in without worries: location privacy in location based social networks. In: 2013 Proceedings IEEE INFOCOM, pp. 3003–3011. IEEE (2013)
7. Carbunar, B., Sion, R., Potharaju, R., Ehsan, M.: Private badges for geosocial networks. IEEE Trans. Mob. Comput. **13**(10), 2382–2396 (2014)
8. Cho, E., Myers, S.A., Leskovec, J.: 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 (2011)
9. Cheng, Z., Caverlee, J., Lee, K., Sui, D.Z.: Exploring millions of footprints in location sharing services. ICWSM **2011**, 81–88 (2011)
10. Li, M., Zhu, H., Gao, Z., Chen, S., Le, Y., Shangqian, H., Ren, K.: 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, pp. 43–52. ACM (2014)
11. Patsakis, C., Zigomitros, A., Solanas, A.: Analysis of privacy and security exposure in mobile dating applications. In: Boumerdassi, S., Bouzefrane, S., Renault, É. (eds.) Mobile, Secure, and Programmable Networking. LNCS, vol. 9395, pp. 151–162. Springer, Switzerland (2015)
12. Rogers, R., Lombardo, J., Mednieks, Z., Meike, B.: Android Application Development: Programming with the Google SDK. O'Reilly Media Inc, Sebastopol (2009)

13. Baidu Inc. Baidu location sdk. http://api.map.baidu.com/lbsapi/cloud/geosdk.htm
14. Murphy, W., Hereman, W.: Determination of a position in three dimensions using trilateration and approximate distances. Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, Colorado, MCS-95, **7**, p. 19 (1995)
15. Lawrence, E.: Fiddler: Web Debugging Proxy (2007)
16. Hickman, K., Elgamal, T.: The ssl protocol. Netscape Communications Corp, vol. 501 (1995)
17. Rudrappa, N.: Defeating SSL certificate validation for android applications
18. Evans, C., Palmer, C., Sleevi, R.: Public key pinning extension for http. Technical report (2015)
19. Nurseitov, N., Paulson, M., Reynolds, R., Izurieta, C.: Comparison of JSON and XML data interchange formats: a case study. Caine **2009**, 157–162 (2009)
20. Winsniewski, R.: Android-apktool: a tool for reverse engineering android APK files (2012)
21. Alll, B., Tumbleson, C.: Dex2jar: Tools to work with android. dex and java. class files
22. Dupuy, E.: Jd-gui: Yet another fast java decompiler. http://java.decompiler.free.fr/?q=jdgui/accessed
23. Berthome, P., Fecherolle, T., Guilloteau, N., Lalande, J.F.: Repackaging android applications for auditing access to private data. In: 2012 Seventh International Conference on Availability, Reliability and Security (ARES), pp. 388–396. IEEE (2012)
24. Android Developers: Using the android emulator (2012)
25. Android Developers: Uiautomator (2013)
26. Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: 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, pp. 50–61. ACM (2012)