On the Privacy of Real-World Friend-Finder Services

Aristide Fattori[†] Alessandro Reina[†] Andrea Gerino[‡] Sergio Mascetti[†‡]
{aristide,ale}@security.di.unimi.it {andrea.gerino,sergio.mascetti}@ew-tech.it
[†]Dept. of Computer Science, Università degli Studi di Milano [‡]EveryWare Technologies

***Abstract*—Privacy protection in the deployment of location based services is a hot topic both in CS research and in the development of mobile applications. In this paper we consider a location based service that currently has hundreds of millions of users and we show how we developed a software that is able to discover their exact positions, by only using information publicly disclosed by the service. Our software does not exploit a specific limitation of the considered service. Rather this contribution shows that there is an entire class of services that is subject to the attack we present.**

I. Introduction

"Friend finders" are popular services that allow a user to discover, through her mobile device, people that are *in the vicinity*. We classify these services into four groups, based on two main technical dimension. First, some friend finders allow each user to know the position of other users, for example showing them on a map. We name these services "explicit position" friend finders[1]. In contrast, "implicit position" friend finders only show location-related information, without providing users' *precise* position[2]. For instance, these services only show users closer than a given distance threshold. The second distinguishing technicality is how users are related to each other: in "closed buddies" services, a user is informed about the position (or related information) of other users in a list of "friends", that must *explicitly* and *mutually* confirm their willingness to be in such a list. For example, when using *Google Latitude*, a user can define the set of other users who are allowed to see her position on the map. Vice versa, in a "open buddies" approach, all users are considered as "friends". For example, *SKOUT* provides a user with the distances from all nearby users.

When using "explicit position" friend finder services, users are well-aware that their position is being publicly disclosed to other users of the service. In contrast, a user of a "implicit position" service would expect her position to be protected from free disclosure to other users. In some "closed buddies" services this is actually the case. For example in *PCube* users have a fine control over the information they disclose to their friends [1]. Overall, most of the scientific contributions addressing this problem consider "closed buddies" services [2], [3], [4], [5].

In this paper we consider commercial friend finder services that are "open buddies" and "implicit position". We show that these services provide a deceitful form of privacy protection. Indeed, while a user's position is not directly transmitted to other users, it is possible to compute the position by elaborating the information that the service provider publicly discloses to any user. In particular, in this paper we consider one of the most popular dating services that uses a friend finder as one of its functionality. The service declares to have more than a hundred of millions of users in total. Since the attacks that we describe may endanger the privacy of the users of this service, we will not report its name but will only refer to it as "the Service". This paper has three main contributions.
1) We describe two different attacks to obtain the position of any user and we give an example of how to perform the attacks manually, i.e., without any ad-hoc software (see Section II).
2) We show how the attacks can be fully automated, through the use of an *ad-hoc* client that can compute, in a few seconds, the position of any user in a given area (see Section III).
3) We describe how the identification of the position of a user in a given area can be used as a primitive to develop even more threatening attacks (see Section IV).

II. Attack description

In this section we first clarify our reference scenario (Section II-A) and then we describe two attacks that disclose the precise location of a target user (Sections II-B and II-C).

A. Scenario definition

A *source* person s is using the Service. Another person t (*target*) is using the same service and is located in the vicinity of s in the sense that t is shown to s as a nearby user. User s is the adversary, as she aims at obtaining the position of t with the highest possible precision. To achieve this, s can collude with one or more buddies $c_1 \dots c_n$. In the following, we denote with s, t and c_i both the actors of the scenario and their positions. We also denote with $d(i,j)$ the distance between two users.

To perform the attack, s relies on the knowledge derived by herself and by the colluding buddies from the use of the service. Also, s can use information about her location and the location of colluding buddies c_i as well as data derived from this, like $d(s, c_i)$.

In this paper we distinguish two attacks. For some target users, the mobile client of the Service shows the distance of the target from the source user. The distance value is approximated to the upper bound of the distance from t, which we denote with $\overline{d}(s,t)$. In this case, we use a "known distances" attack to retrieve the position of t. In other cases the client does not show the distance of the target from the source user. We call the attack in this case the "unknown distances" attack.

B. "Known distances" attack

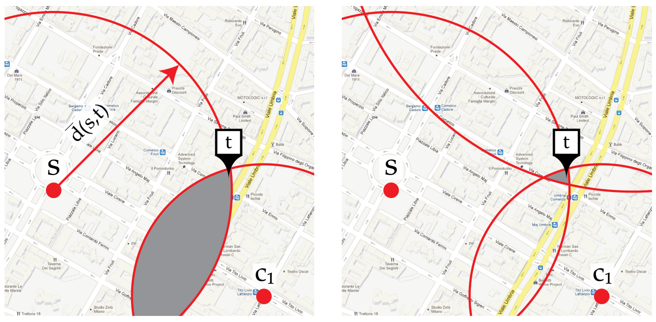
Given the upper bound of the distance $\overline{d}(s,t)$, s can derive that t is located in the circle centered in the position of s with

[1]Examples are "Find my friend", and "Google Latitude".

[2]Examples are *PCube*, *SKOUT*, and *Badoo*.

radius $\overline{d}(s,t)$. Clearly, if s also knows $\overline{d}(c_1,t)$ for a colluding buddy c_1, then it is possible to further restrict the position of t to the intersection of the two circles (see Figure 1(a)). This is similar to a trilateration attack [6], with the main difference that the exact distance is unknown. If there are more colluding buddies, the position of t can be identified with less approximation (see Figure 1(b)).

Although the above attack is straightforward from a theoretical point of view, a number of issues could arise in its practical application: errors due to GPS, approximations in the server-side distance computation, delays in the service provisioning, and so on. To evaluate the feasibility of this attack in practice, we kept the target user in a fixed position and we used several observations from a moving source user s to simulate a set of colluding buddies. In our experience with the Service, three observations are sufficient to locate t with a good approximation. For example, in Figure 1(a) t is located in an area of less than 0.05km^2 while in Figure 1(b) the area is about 500m^2.



(a) Two observation points (b) Three observation points

Fig. 1. "Manual" execution of the "known distances" attack.

C. *"Unknown distances" attack*

When the distance of the target from the source user is unknown, it is not possible to directly compute $\overline{d}(s,t)$. However, we now show how this value can be derived by exploiting the fact that the client shows to the user s the list L of nearby users, ordered according to their distance from s.

In this case, s can discover the approximate distance to t by colluding with another user c as follows: c starts from s moving away from this position, while s periodically monitors L as well as the distance between s and c. As long as c precedes t in the list of users, s knows that c is closer than t. When c happens to be after t in L, then $d(s,t) < d(s,c)$. Since $d(s,c)$ is known, s actually discovers $\overline{d}(s,t)$. Once the approximated distance is discovered, the "known distance" attack can be used to discover the position of t. Note that in this attack we are implicitly assuming that t is not moving during the time of the attack. In Section III we show that this assumption is not necessary while performing the automated attack.

Example 1. *At time $T=0$, c is located in the same position as s, hence c is the first element of L (see Figure 2(a)). At time $T=1$, c has moved at a distance of $250m$ from s, but still precedes t in L (see Figure 2(b)). At time $T=2$, c is shown in L after t and the distance between c and s is $280m$ (see Figure 2(c)). The adversary concludes that the distance between s and t is between $250m$ and $280m$ and hence t is located in the gray area of Figure 2(d).*

III. Attack Automation

The attack we illustrate in Section II is conducted by a human agent by interacting *manually* with the Service. Manual interaction, however, greatly undermine the scalability of the attack. Even assuming that the human user has the ability to feed *false* location information to the Service (i.e., she must not *physically move* to different real-world locations during the attack), manually performing every step needed during the attack may be cumbersome. For this reason, we investigated how to automate the attacks to our target service. To achieve this, we developed a software agent that automatically communicates with the service provider, pretending to be a mobile client in use by a real user.

A. *Development of ad-hoc client*

To develop an ad-hoc client it is first necessary to figure out how a real client communicates with its service provider. To this end, we installed the application on an Android 2.2 system, running inside the Android Emulator [7] and we configured it to connect to the Service with a user that we had previously registered. Then, using the Android SDK functions we "placed" the device in a geographical location and we used the client to update the location and to retrieve nearby users. We repeated this operation several times, each time "placing" the device in a different position. While doing this we captured the network traffic produced by the device and we analyzed it to understand the communication protocol.

By observing the network traffic we identified a known, non-textual, protocol used to efficiently exchange marhsalled data over a network connection. Since this protocol makes it possible to define *ad-hoc* data types, we created a custom parser to correctly identify different messages. Eventually, we identified most of the messages and we also realized that, for some requests, the service provider does not require authentication, making it possible to obtain important information without registering any user.

After understanding the communication protocol, we developed a Python application capable of communicating with the service provider to compute the following primitive:

$$\mathcal{U} = getNearby(lat,\ lon,\ \delta)$$

The primitive takes as input a geographical location $\langle lat,\ lon\rangle$ and a distance δ, returning a set $\mathcal{U}$ of users reported by the Service as located at most at distance δ from $\langle lat,\ lon\rangle$.

We observed that the Service does not adopt any security measure, such as encryption, to protect the network traffic generated by its users while using the client. Adopting such a solution, for example by migrating the communication protocol over HTTPS, would undoubtedly increase the overall security of the service, for example preventing an external adversary from sniffing the network traffic. However, note that this is not a limitation of our attack. Indeed, there are many ways we could still retrieve the information we need about the communication protocol. A clever attacker, for example, can reverse engineer the target application to view the source

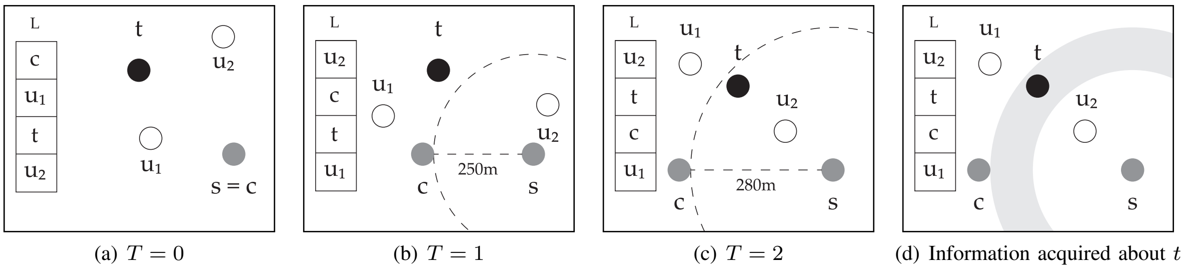


Fig. 2. Example of "unknown distances" attack.

code responsible for the communication protocol. Otherwise, if the application or its user fails to properly validate the SSL certificate, a Man-in-the-Middle [8] attack can be conducted to trick the application into using a fake certificate, customly created by the attacker that can, consequently, decrypt `HTTPS` traffic. We adopted the latter technique to understand the communication protocol of two friend finder services.

B. Attack Algorithm

While developing the ad-hoc Python client, we noticed how data exchanged through the Service's client and server include the *precise* distance between the client's position and nearby users. Thus, we customized the *getNearby()* function to return such piece of information too.

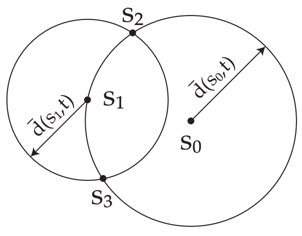


Fig. 3. Source points chosen by the automated attack.

When precise distances between users are known, the location of the target can be obtained with trilateration. First, we use the *getNearby()* function from a source position s_0 to get the list of nearby users among which we chose the target t. Since the service provides precise distances, we acquire the value $d(s_0, t)$. We then choose a point s_1 on the circle centered in s_0 with radius $d(s_0, t)$ (see Figure 3). Again, we use *getNearby()* to retrieve $d(s_1, t)$. Now, let s_2 and s_3 be the two intersections between the two circles centered in s_0 and s_1 with radius $d(s_0, t)$ and $d(s_1, t)$, respectively. We use *getNearby()* for the third time to compute $d(s_2, t)$: if the result is close to zero, then we conclude that t is close to s_2, otherwise t is close to s_3.

This algorithm has the advantage of being simple from a conceptual point of view and to require a constant number of executions of the *getNearby()* primitive; hence, it has a short execution time (a couple of seconds, in our experiments, mainly due to network latency). Also, the position of t can be obtained with high precision. In our experiments, the average error is in the order of a few meters (always less than 10m).

IV. Privacy Implications

As shown is Section III it is possible to automate the privacy attacks described in Section II to discover the position of a target user t under the assumption that t is located close to the source user s. In this section we show how this can be used to achieve three threatening privacy attacks.

A. "Who is there?" attack

The aim of the "Who is there?" attack is to understand who resides in a given location. Intuitively this attack is particularly threatening when the presence in the chosen location discloses personal data about the user. For example if the adversary chooses a place of worship as target location she can infer, with a certain likelihood, the religious belief of the people at that location. Repeating the observation and checking who is present in that location several times can increase the probability of a correct guess. Technically, the attack can be simply performed by positioning s at the target location and retrieving the users that are closer than a given threshold distance with the *getNearby()* primitive.

B. "Where is Alice?" attack

Let us consider an adversary that wants to stalk a target user t. Since the approximate position of t is unknown, we cannot directly use the automated attacks presented in Section III because we do not know where to place s_0. In practice, we first need to find a position s_0 such that t is "near-by". Then, we can use the attacks shown in Section III.

To find the position of s_0, we iteratively use the *getNearby()* primitive to "search" for t. In theory, this can be achieved by starting from a given random position and retrieving the nearby users. If t is not in the result, the adversary can chose a new random source position retrieving nearby users to that point. Eventually the location of the target user is found. Clearly, several optimizations are possible. in area where t has already been searched.

In practice, such an attack requires to issue a large number of requests and to retrieve a number of users linear in the total number of users of the service. In our automated attack, we observed that it is possible to retrieve from the service provider about 250 users per second. This means that searching t in a million of users takes about one hour. If the service has hundreds of millions of users this attack is impractical, unless we have some clues about t like profile information (that can be used as filters) or the region where t is likely to be located. For example, considering only female users aged between 20 and 25 years, we have been able to retrieve users in an area of 13km centered in Milan in about half a second.

C. "Follow Alice" attack

By periodically repeating the "Where is Alice?" attack and storing the results it is possible, after some time, to identify the set of places visited by a target user t. From this "trace", the attacker can discover t's home address and workplace and potentially spot t's real identity.

Clearly the "trace" of places visited by t contains only the locations sent to the service providers by t's client. Most of the services currently available (including the Service) use clients that send location updates in response to user-initiated actions like log-ins, searches for nearby users or explicit requests. However, some clients allow the user to enable automatic location updates hence periodically sending the user's location to the service provider, in some cases even when the application is in background. This is a more threatening situation, since a user can be unaware of being disclosing her position.

V. Ethical Considerations

Our purpose in this work is to demonstrate *how* an attacker can leverage *publicly available* information provided by a commercial friend finder service in order to *precisely* infer the position of an arbitrary and unaware user. In our attack we do not violate or hack any system. Actually, our objective is not to show security vulnerabilities in the considered services, but rather to show that it is possible to create an application that pretends to be a client and use it to automate privacy attacks. More specifically, our privacy attack is performed according to the following principles.
1) The attacker does *not* anyhow compromise the servers of the service provider or retrieve otherwise unavailable information.
2) The attacker does *not* interact with her target of choice (e.g., tricking him to visit a specially crafted web page).
3) Every information that the attacker uses to infer the position of her target(s) is available either to registered or unregistered users of the service.
This being said, when performing the experiments that are required to proof the soundness of our work, the privacy of real users of the Service must be taken into high consideration. To this end, during our analyses, we targeted only users under our direct control and users of whom we had previously got an explicit authorization.

VI. Conclusions

In this contribution we have shown that users participating in a "open-buddy" friend finder expose their locations to the public, even if this information is not explicitly given to other users. Indeed, we showed that, after creating an ad-hoc client, it is relatively easy to use public information to spot a user's positions and even to follow a target. While we have developed our ad-hoc client for a desktop computer, it would be possible to run similar code on a mobile device, enabling accessible "on the move" attacks.

The defense against the attack we presented is non trivial. Technically, this is due to the fact that in an "open-buddies" friend finder some location-related information need to be disclosed to the public and the malicious use of this information can easily lead to discover the actual position of the target user. So far, we did not devise any security-related solution to prevent the adversary from learning the communication protocol and actually we have been able to understand and replicate all the protocols of the many service that we investigated. Probably a partial solution to the problem would consist in rendering the attacks more complicated by identifying the attack patterns (e.g., series of requests) and blocking them. However this can hardly be a general solution to the problem.

Similarly, we have not been able to identify any data-management solution to prevent these attacks. One approach that partially enhances users' privacy is to disclose only approximate position (like the ZIP code, for example). Some services actually implement this solution. While this information is intuitively less sensible than the exact location, disclosing it still causes some privacy issues. Another limit of this solution is that it trades-off privacy for computed distance precision, causing a decrease in the quality of the service.

We have one last consideration about the users' perception of the above problems. We created and published a non-technical video to briefly present the above results to end-users[3]. The video was seen by less than 400 persons. We identified two reasons for this unsatisfying result. First, despite our effort, we had not been able to advertise the video to the correct audience or to make it sufficiently clear. Second, we collected comments showing that apparently people do not have the perception of how much their privacy in endangered by automated attacks. While this is not a strictly technical problem (rather it is a sociological one) we argue that it should be taken into account while devising privacy-preserving solutions.

Acknowledgments

This work was partially supported by Italian MIUR under grant FIRB RBFR081L58_002.

References

[1] S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia, "Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies," *The VLDB Journal*, vol. 20, no. 4, 2011.

[2] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, Lester and Pierre: Three protocols for location privacy," in *Privacy Enhancing Technologies*, vol. LNCS 4776. Springer, 2007, pp. 62–76.

[3] L. Šikšnys, J. R. Thomsen, S. Šaltenis, M. L. Yiu, and O. Andersen, "A location privacy aware friend locator," in *Proc. of the 11th Int. Symposium on Spatial and Temporal Databases*, ser. LNCS. Springer, 2009.

[4] L. Šikšnys, J. R. Thomsen, S. Šaltenis, and M. L. Yiu, "Private and flexible proximity detection in mobile social networks," in *Proc. of the 11th Int. Conf. on Mobile Data Management*. IEEE Comp. Soc., 2010.

[5] S. Mascetti, C. Bettini, and D. Freni, "Longitude: Centralized privacy-preserving computation of users' proximity," in *Proc. of 6th VLDB workshop on Secure Data Management*, ser. LNCS. Springer, 2009.

[6] H. L. Groginsky, "Position estimation using only multiple simultaneous range measurements," *Aeronautical and Navigational Electronics, IRE Transactions on*, vol. ANE-6, no. 3, pp. 178 –187, sept. 1959.

[7] "Android SDK," http://developer.android.com/sdk/index.html.

[8] A. Ornaghi and M. Valleri, "Man in the middle attacks," in *Blackhat Conference Europe*, 2003.

[3]http://watchyourstep.everywaretechnologies.com/