Where's Wally? Precise User Discovery Attacks in Location Proximity Services

Iasonas Polakis   George Argyros   Theofilos Petsios
Suphannee Sivakorn   Angelos D. Keromytis

Network Security Lab, Computer Science Dept.
Columbia University, New York, NY, USA
{polakis, argyros, theofilos, suphannee, angelos}@cs.columbia.edu

Abstract

Location proximity schemes have been adopted by social networks and other smartphone apps as a means of balancing user privacy with utility. However, misconceptions about the privacy offered by proximity services have rendered users vulnerable to trilateration attacks that can expose their location. Such attacks have received major publicity and, as a result, popular service providers have deployed countermeasures for preventing user discovery attacks.

In this paper, we systematically assess the effectiveness of the defenses that proximity services have deployed against adversaries attempting to identify a user's location. We provide the theoretical foundation for formalizing the problem under different proximity models, design practical attacks for each case, and prove tight bounds on the number of queries required for carrying out the attacks. To evaluate the completeness of our approach, we conduct extensive experiments against popular services. While we identify a diverse set of defense techniques that prevent trilateration attacks, we demonstrate their inefficiency against more elaborate attacks. In fact, we pinpoint Facebook users within 5 meters of their exact location, and 90% of Foursquare users within 15 meters. Our attacks are extremely efficient and complete within 3-7 seconds. The severity of our attacks was acknowledged by Facebook and Foursquare, both of which have followed our recommendations and adopted spatial cloaking to protect their users. Furthermore, our findings have wide implications as numerous popular apps with a massive user base remain vulnerable to this significant threat.

1. INTRODUCTION

Location-based services (LBS) have become an integral part of everyday life. However, accessibility to fine-grained location information has raised significant privacy concerns, as users are exposed to various threats, ranging from the inference of sensitive data [33] (e.g., medical issues, political inclination and religious beliefs) to physical threats such as stalking [10]. Furthermore, apart from the revelations regarding mass user surveillance by government agencies, articles have revealed that law enforcement agencies also follow more targeted, and unorthodox, tactics. Fake profiles are used to befriend users and gain access to personal data, as well as track their whereabouts by monitoring their check-in behavior [6, 8]. Therefore, the information accessible by users' contacts is a significant aspect of their privacy.

Revealing a user's location is considered a significant privacy breach [46], and services are adopting the more privacy-preserving approach of *location proximity*: notifying users about who is nearby, and at what distance. However, when the exact distance to a user is revealed by the service, trilateration attacks become feasible, with several examples being presented in the media recently. Articles have also reported that the Egyptian government used trilateration to locate and imprison users of gay dating apps [7,9]. While the use of trilateration has not been confirmed, such reports highlight the potential severity of such attacks, and the importance of preserving the locational privacy of users. Naturally, these reports have caught the attention of popular services, which in turn have deployed defense mechanisms to prevent localization attacks [2].

In this paper, we explore the privacy guarantees of 10 popular social networks and LBS. We audit the services and identify the mechanisms deployed to protect the location privacy of their users. To evaluate the defenses that have been adopted by the industry, we formalize the problem of locating users as a search problem in the discrete Euclidean plane. To our knowledge, this is the first formal treatment of user discovery attacks in proximity services. We prove tight bounds on the number of queries required to attack a service under different proximity models, and devise optimal algorithms that realize those attacks. The lower bounds on the query complexity of our techniques provide useful insight on the effectiveness of mitigations against localization attacks, such as rate limiting the number of queries.

We evaluate our attacks against four of the audited services that employ a diverse set of countermeasures. We show that user discovery attacks against proximity services may require complex techniques; our attacks include geometric algorithms that gradually reduce the candidate bounding area where a user resides, the employment of colluding accounts for obtaining side channel information on the distance between users, and the utilization of statistical algorithms for coping with the randomization used by services as a defense mechanism. Our results demonstrate that, despite the defense mechanisms in place, our attacks are still very effective and time-efficient, and practical for use at scale and on a continuous basis (real-time tracking). In particular, using a single account, we pinpoint Facebook users within 5 meters of their actual location in 3 seconds, and 90% of Foursquare's Swarm users within 15m in 7 seconds. We even stress-test our attacks and demonstrate the feasibility of tracking moving targets in real time. Due to the recent events [9], Grindr hides the distance information for citizens of oppressive regimes. Even without any distance information disclosed, we are able to carry out successful attacks by inferring the distance to our target. Using a pair of colluding accounts, and the distance-based ordering of users by Grindr, we pinpoint 67% of the users within 10m of their exact location, and 98% within 19m. Similarly, even though Skout implements a sophisticated randomization defense, we are able to pinpoint its users within 37.4m on average.

Our findings reveal that there is no industry standard for ensuring the locational privacy of users; attempts are based on ad-hoc approaches that often exhibit a lack of understanding of the technical intricacies of localization attacks. Despite the active effort to prevent such threats, every service we audited was vulnerable to, at least, one of our attacks. To provide a robust solution, we revisit an obfuscation mechanism from the literature, namely spatial cloaking [19], and apply it to the domain of distance-based proximity services. By quantizing the plane and mapping users to points on a grid, the service can prevent adversaries from pinpointing users to a finer precision than that of a grid cell. To incentivize services to adopt this defense, we provide a precise characterization of both the privacy obtained (under certain assumptions), and the tradeoff between privacy and usability. *After our disclosure, Facebook and Foursquare acknowledged the severity of our attacks and, following our guidelines, adopted spatial cloaking for protecting their users.*

The main contributions of this paper are:

- We present a formal treatment of user discovery attacks within a proximity service. We model the problem, prove the lower bounds on the query complexity, and design algorithms that match the corresponding lower bounds.
- We evaluate the privacy of popular proximity services through extensive experimentation, and reveal the shortcomings of existing proximity models and defenses. The disclosure of our findings to the services resulted in Facebook and Foursquare adopting spatial cloaking.
- We analyze the practical aspects of our attacks, and identify key characteristics that affect their performance and accuracy. We provide guidelines for impairing the attacks and ensuring a minimum level of privacy without incurring a significant deterioration of the quality of service.
- We release an open-source auditing framework for assisting developers and researchers in assessing the privacy of proximity services. Our framework has already been used by Facebook for evaluating their newly-adopted spatial cloaking mechanism.

2. MODELLING DISCOVERY ATTACKS

In this section we provide the theoretical modelling of our user discovery attacks. For simplicity, we refer to the adversary as Mallory and the target user as Wally.

Threat Model. The adversary can be any entity interested in determining a user's location; a government or law enforcement agency conducting user surveillance ([6, 8]), a third party (e.g., insurance company) interested in inferring private data or a malicious individual (e.g., stalker) [10]. To highlight the inefficiency for existing designs and countermeasures, we adopt a weak adversarial model: the adversary uses only the distance information revealed by the service.

Our attacks *do not require prior knowledge of the user's whereabouts*, and the only requirement is to have an account in the service so as to obtain some type of information about the distance to the user. In Section 5 we demonstrate that we can identify a user's location with high precision, and also track a moving target in real time.

Problem Formulation. We formulate our problem as a search problem in the discrete Euclidean plane. This is justified by the fact that both services and protocols (e.g., GPS) cannot provide arbitrary accuracy. By modelling it as a discrete problem, we can adapt the size of the input to match the accuracy provided by the service.

We consider a target user u residing at a point p_u of the discrete Euclidean plane. The attacker can request proximity information regarding the location of the user u. This is obtained through an oracle, which we refer to as a *proximity oracle* P. Since the attacker can fake her own location, she can query the proximity oracle from any point within the Euclidean plane. Thus, the proximity oracle accepts a point p and returns proximity information for the point p and the location p_u of the target user. We denote by $P_u(\cdot)$ the proximity oracle which, for an input of a point p, outputs some function of p, p_u. Also, we define as $\text{dist}(p_1, p_2)$ the Euclidean distance between two points p_1, p_2. We proceed to define the user discovery problem, our main algorithmic problem, in the context of location proximity services.

Definition 1. ***User Discovery Problem (UDP):*** *Let p_u be a point in the discrete Euclidean plane and A an area containing p_u. In the User Discovery Problem the goal is to identify the point p_u, given as input the area A and black box access to a proximity oracle P_u.*

In the following sections we will describe three different implementations of the proximity oracle that capture the protocols used by real services. For each of these oracles, we describe how to solve UDP given access to the respective oracle.

2.1 Disk User Discovery Problem

We start by giving the definition of the first oracle.

Definition 2. ***Disk Proximity Oracle:*** *A disk proximity oracle $P_{r,u}(p)$ with radius r, accepts as input a point p in the discrete Euclidean plane and is defined as:*

$$P_{r,u}(p) = \begin{cases} 1 & \textit{if } \text{dist}(p, p_u) \leq r \\ 0 & \textit{otherwise} \end{cases}$$

This model captures services and protocols that inform the user whether another user is within a certain distance of his current location; otherwise the user is not in proximity and no further information is given. We define the Disk User Discovery Problem (DUDP) to be the UDP given black box access to a Disk Proximity Oracle. We solve DUDP by partitioning the problem into two subproblems, which require a different approach in order to be solved: first, we wish to restrict the user within a single disk of radius r and, second, to search that disk for the target point p_u.

In the former subproblem, the user is given a, possibly large, area A which she wants to cover with disks of radius r in order to restrict the search area within a single disk. We call this problem the Disk Coverage Problem. To achieve an efficient attack, we wish to cover the area with the minimum number of disks.

Definition 3. *In the* ***Disk Coverage Problem****, the input is an area A in the discrete Euclidean plane and a number $r > 0$. The goal is to cover the area A with the minimum number of disks of radius r.*

After the target user's location is restricted within a single disk of radius r, one has to use the proximity oracle to further refine the user's location up to a single point. We call this subproblem the Disk Search Problem.

DEFINITION 4. *In the **Disk Search Problem** the input is a single disk of radius r along with a proximity oracle $P_{r,u}(\cdot)$. The goal is to uniquely pinpoint the point p_u within the input disk.*

Notice that the Disk Search Problem is exactly the DUDP when the input area is restricted to a disk of radius r. Because the two cases are handled in a different manner, we address them separately. Next, we examine each subproblem and describe algorithms for solving them.

Solving Disk Coverage. To generalize our attack, we assume that the only information the attacker has is a very coarse-grained approximation of the location of the targeted user; for example Mallory might know which state Wally lives in. Given a total area in which the user might reside, our first goal is to to pinpoint the user within a disk of radius r, as provided by the proximity oracle.

A problem that corresponds precisely to the Disk Coverage Problem is the Minimum Dominating Set (MDS) problem in a special class of graphs called Unit Disk Graphs (UDG). In the MDS problem, one is given as input a graph $G = (V, E)$ and the goal is to find a set $D \subseteq V$ such that for every $v \in V$ there exists a $u \in D$ for which $(u, v) \in E$. UDG are a special class of geometric graphs; even though a number of equivalent definitions exist, we will use what is referred to as the proximity model [15]:

DEFINITION 5. *(Proximity model for UDG) Consider a set P of n points in the plane. The **Unit Disk Graph** for the points in P is the undirected graph $G = (P, E)$ where P is the set of n points and for every $u, v \in P$, we have that $(u, v) \in E$ iff $\text{dist}(u, v) \leq k$, for some predefined $k > 0$.*

The MDS problem in UDG has been studied extensively. While it is NP-Hard [32], the approximation ratio that has been achieved is much better than for the general MDS problem which cannot be approximated better than $O(\log n)$. Specifically, there exists a 5-approximation in linear time [27], a 3-approximation in time $O(n^{18})$ [16] and a PTAS [35] whose complexity is impractical for the problem instances we aim to tackle.

Notice that the areas that one needs to cover might spread across entire cities or even larger areas. Thus, the respective graphs may contain millions of nodes. At that scale, even algorithms with quadratic complexity may result in prohibitively large running times. Fortunately, for the MDS problem there exists a straightforward, linear-time algorithm.

The algorithm works by taking an arbitrary node from the graph, adding it to the dominating set, removing all neighboring nodes that are already covered by the chosen node, and repeating the process until the graph is empty. It is easy to see that the algorithm will run in time $O(|V|)$. If the graph is a UDG, the algorithm provides a covering with a size at most 5 times the minimum covering [27]. We create the points of our set by tiling the 2-dimensional plane with hexagons, a common technique for ensuring coverage with minimum node placement in cellular networks [43]. Each hexagon edge is equal to $\sqrt{3}r$, where r is the radius for the lookup granularity. While the construction is heuristic, since no two points in our set are at a distance less than r, our algorithm creates a 5-approximation of the minimum covering, and constructing the points takes $O(|V|)$ on average.

S	Input set of points to `DiskSearch`$(\cdot)$
$R(S)$	Circumscribed rectangle of set S
k	Length of the short edge of $R(S)$
d	Length of the long edge of $R(S)$
$I_{R(S)}$	Set of points within $R(S)$
$C_r(p)$	A disk centered at point p with radius r
p_m	Middle point of the short edge of $R(S)$
l_m	Line parallel to the long edge of $R(S)$ and crossing p_m
S_1	$S \cap C_r(p)$
S_2	$S \setminus S_1$

Table 1: Notation used for the `DiskSearch`$(\cdot)$ algorithm.

Solving Disk Search. After the user is restricted to a disk of radius r, the goal is to refine the location to a single point. We present an algorithm that solves the Disk Search problem using $O(\log r)$ queries. One can easily show that the result is optimal. Our algorithm works like a binary search algorithm, using the oracle to cut down the candidate points that the user resides in by half with each query. Notice, that we might not always be able to split the candidate points into two equals sets in every cut, however, as we will show, the overall number of queries required is still $O(\log r)$.

We start by providing some definitions that will be used throughout the algorithmic description: for a given set S of points in the Euclidean space, we denote with $R(S)$ the circumscribed rectangle that includes those points, and by $I_{R(S)}$ the points in the rectangle $R(S)$. The two edges of $R(S)$ are referred to according to their lengths, namely, "short edge" and "long edge" with lengths k, d where $k \leq d$. For a point p, we denote with $C_r(p)$ the points in the disk centered at p with a radius r. We say that the algorithm performs a cut on a set S, resulting in two subsets S_1, S_2, when the proximity oracle $P_{r,u}$ is queried on a point p and $S_1 = S \cap C_r(p)$, $S_2 = S \setminus S_1$. We denote the middle point in the short edge of $R(S)$ by p_m. Finally, let l_m be a line parallel to the long edge of $R(S)$ passing through p_m. Table 1 provides an easy reference of our notations.

The algorithm `DiskSearch`(S) proceeds recursively:

1. For input of a set S, if $|S| = 1$ return $p \in S$.
2. For each point p_i in l_m, starting at a distance r from $R(S)$, and moving towards $R(S)$, check the size of the set $S \cap C_r(p_i)$. If for a point p_i it holds that $|S \cap C_r(p_i)| > |S|/2$, then set $p = p_{i-1}$.
3. Make a call to the proximity oracle on point p:
   - If $P_{r,u}(p) = 1$ recurse on S_1.
   - If $P_{r,u}(p) = 0$ recurse on S_2.

The correctness of the algorithm is evident once we bound the number of recursive calls that the algorithm makes. For the running time analysis we assume that the calls to $P_{r,u}(\cdot)$ require constant time. We prove the following theorem.

THEOREM 1. *The `DiskSearch`$(\cdot)$ algorithm has a time complexity of $O(r \log r)$, and will do at most $O(\log r)$ queries to the proximity oracle $P_{r,u}(\cdot)$.*

We will start by proving a number of useful lemmas:

LEMMA 1. *Let S be a set of points, k be the short edge of $R(S)$ and p, p' two successive points on l_m. Furthermore, let $S_1 = S \cap C_r(p)$ and $S'_1 = S \cap C_r(p')$. Then*

$$|S'_1| \leq |S_1| + O(k)$$

PROOF. The number of points that are added into the disk by shifting the cut by one point are, at most, proportional to the length l of the arc segment with chord length k. We will show that the length is $O(k)$.

First, notice that the length of the arc is less than the perimeter of the circumscribed rectangle around the arc segment. Therefore, if h is the height of the arc segment then we have that $l \leq 2k + 2h$. We want to bound the height h to be less than k. Therefore, we have:

$$\begin{array}{rcl} h & \leq & k \\ \Rightarrow \quad r - \sqrt{r^2 - k^2/4} & \leq & k \\ \Rightarrow \quad k & \leq & \frac{8r}{5} \end{array}$$

If $k > \frac{8r}{5}$, then we have that $l \leq \pi \cdot r \leq O(k)$ therefore the lemma holds trivially. On the other hand, if $k \leq \frac{8r}{5}$ we have that $l \leq 2k + 2h \leq 4k \leq O(k)$.

□

To facilitate computations, we work with the circumscribed rectangles of the sets that result from the cuts of the algorithm. The next lemma shows that the number of points within the sets resulting from cuts and the number of points in their circumscribed rectangles are of the same order.

LEMMA 2. *Let S be the input set in any iteration of the algorithm. Then we have that $|S| = \Theta(|I_{R(S)}|)$.*

PROOF SKETCH. We will show that if the point sets that are input to the algorithm satisfy the invariant before the cut, then the new sets will also satisfy the invariant.

- **Base Case:** The first input is all the points in a disk, which is of the order of $\pi \cdot r^2$. The circumscribed polygon (in this case a square) contains $(2r+1)^2$ points, therefore clearly in this case we have that $|S| = \Theta(I_{R(S)})$.
- **Inductive Step:** Assume now that we are at some iteration where the input set satisfies the invariant.

  Consider the short edge of $R(S)$ with length k. Then, $|I_{R(S)}| = \Omega(k^2)$. Moreover, because $|S| = \Theta(|I_{R(S)}|)$ by the inductive hypothesis we have that $k = O(\sqrt{(}|S|)$.

  Let S_1, S_2 be the two subsets after the cut. Then by lemma 1 we have that

  $$|S|/2 \leq |S_1| \leq |S|/2 + O(k) \leq |S|/2 + O(\sqrt{|S|})$$

  Therefore, $|S_1| = \Theta(|S|)$. In addition, we have that

  $$|S|/2 \leq |I_{R(S_1)}| \leq |I_{R(S)}| \leq O(|S|)$$

  where the last inequality also holds from the inductive hypothesis. Consequently, $|I_{R(S_1)}| = \Theta(|S|)$ and thus, $|S_1| = \Theta(|I_{R(S_1)}|)$ which concludes the proof for S_1. The proof for the set S_2 is similar and, thus, omitted.

□

The next lemma states a bound on the size of the cuts performed by the algorithm, and is basically proven in the inductive step of lemma 2. Thus, we omit its proof.

LEMMA 3. *Let $S_l \in \{S_1, S_2\}$ be the largest subset of S after a cut of the algorithm. Then we have that $|S|/2 \leq |S_l| \leq |S|/2 + O(\sqrt{|S|})$.*

We now proceed with the proof of theorem 1:

PROOF OF THEOREM 1. Notice that since our initial set S contains all the points in a disk of radius r, the number of points that one should test during the linear sweep is at most r at each iteration. In addition, since each iteration makes one query to the proximity oracle, all that remains is to bound the number of iterations of the algorithm.

Using lemma 3 we can bound the total number of queries as follows; We have that queries of the algorithm are bounded by the following recursive equation:

$$T(n) = T(n/2 + K(n)) + 1$$

for some function $K(n)$. Notice now that, because $K(n) \leq O(\sqrt{n})$, there exists an integer c such that, for all $n > c$, we have that $K(n) \leq n/4$. Therefore, for all $n > c$ we have that

$$\begin{array}{rcl} T(n) & \leq & T(n/2 + n/4) + 1 \\ & \leq & O(\log n) \end{array}$$

When $n \leq c$ the algorithm would need a constant number of additional queries and, therefore, the total number of queries to the proximity oracle is at most $O(\log n)$, which concludes theorem 1.

□

Moreover, we prove the following theorem showing the optimality of our algorithm.

THEOREM 2. *Let $P_{r,u}(\cdot)$ a proximity oracle with radius r, and A be a set of points in the discrete Euclidean plane. Denote by OPT_{MDS} the minimum size of a dominating set for the Unit Disk Graph spawned from the points in A. Then, any deterministic algorithm solving the DUDP in the set A will have to make $\Omega(OPT_{MDS} + \log r)$ queries to the disk proximity oracle $P_{r,u}(\cdot)$ in the worst case.*

We prove the theorem by first showing a lower bound for the Disk Search problem.

LEMMA 4. *Any algorithm solving the Disk Search problem has to make $\Omega(\log r)$ queries to the proximity oracle $P_{r,u}(\cdot)$.*

PROOF. Consider the binary decision tree of the algorithm based on the results of the proximity oracle queries. Then because there are $\Omega(r^2)$ possible points where the user might reside, the tree must have $\Omega(r^2)$ leaves and, therefore, the height of the tree must be at least $\Omega(\log r)$. □

Now we proceed with the proof of Theorem 2.

PROOF OF THEOREM 2. In the case that $OPT_{MDS} \geq \Omega(\log r)$ we have that, in the worst case, at least OPT_{MDS} queries will be needed in order to cover the area and we are done. In the opposite case that $OPT_{MDS} \leq O(\log r)$, one has to at least search a circle of radius r which as, we showed above, cannot be done in less than $\Omega(\log r)$ queries. □

2.2 Rounding User Discovery Problem

Now we turn to a different oracle implementation. This proximity oracle model computes the distance between p and p_u but rounds the resulting value up to some predefined accuracy. To correctly define the oracle in this model, we define the notion of a rounding class.

DEFINITION 6. *(Rounding class) For an interval $I_k = (a, b) \subseteq \mathbb{R}_+$ we define the rounding class R_k to be the tuple (I_k, δ_k) where $\delta_k \in \mathbb{R}_+$ is a rounding value.*

Algorithm 1: Algorithm for the RUDP

```
Input: Rounding Proximity Oracle $P_{S,u}(\cdot)$, Area $A$;
Output: Location of the target user in the Euclidean plane;
$\texttt{Area}_0 = A$;
$i = 1$;
$l = |S|$;
while $i < l \wedge |Area_i| > 1$ do
    $\texttt{Area}_i = \texttt{Trilaterate}(\texttt{Area}_{i-1}, P_{S,u}(\cdot))$;
    $i = i + 1$;
end
$P_{r,u} = \texttt{SimulateDiskProximityOracle}(P_{S,u}(\cdot))$;
$p_u = \texttt{DiskSearch}(\text{Area}_k, P_{r,u}(\cdot))$;
```

Some services (e.g.,Facebook) apply different rounding depending on the distance between two users. This motivates the notion of a rounding class family, which we define below so as to capture this behavior.

Definition 7. *(Rounding Class Family) We define a family of rounding classes $S = \{(I_1, \delta_1), \ldots, (I_n, \delta_n)\}$ to be a set of rounding classes such that the following hold:*
1. The set $I = \{I_1, \ldots, I_n\}$ forms a partition of $\mathbb{R}_+$.
2. For i, j we have that $i < j \Rightarrow \delta_i < \delta_j$.
3. For $k > 1$, it holds that $\delta_k \leq \inf I_k$.

The first condition implies that for every possible distance there is a corresponding rounding value. The second condition asserts that the rounding values monotonically decrease as the distance decreases. The third condition intuitively states that the rounding added is small compared to the actual distance. Conditions 2,3 are not necessary for our attacks to work; however they are natural constraints which we found in all services, and simplify the description and analysis of our algorithms.

Next, we define the operator $\lceil x \rceil_\delta$ that rounds x to the closest number $x' \geq x$ such that $\delta \mid x'$[1]. Notice that, one can similarly define the operators $\lfloor \cdot \rceil_\delta$ and $\lfloor \cdot \rfloor_\delta$. Indeed, when attacking real services, we consider rounding classes that also use these operators. However, the algorithms themselves does not change in any way and. For simplicity, we will use the $\lceil \cdot \rceil_\delta$ operator throughout the presentation and analysis.

Definition 8. *A **Rounding Proximity Oracle** $P_{S,u}(p)$ indexed by a family of rounding classes S is defined as: Let $(I_k, \delta_k) \in S$ be a rounding class such that $\text{dist}(p, p_u) \in I_k$. Then, $P_{S,u}(p) = \lceil \text{dist}(p, p_u) \rceil_{\delta_k}$.*

We define the Rounding User Discovery Problem (RUDP) as the UDP given access to a Rounding Proximity Oracle. Our problem definition also covers other models. E.g., an oracle that outputs exact distances is a rounding proximity oracle with the rounding class family $S = \{(\mathbb{R}_+, \delta)\}, \delta \to 0$.

Solving RUDP. Intuitively, if we have the exact distance to the target user, a simple trilateration will give us his location. However, in our case, the rounding operation is applied, which adds noise to the result. Thus, the result of applying a trilateration on a rounded distance is to reduce the area where the user resides. Next, we exploit the fact that for each rounding class $R_k = (I_k, \delta_k)$ we have that $\delta_k \leq \inf I_k$. Reapplying the trilateration algorithm within the reduced area will result in getting smaller rounding errors and, thus, further reducing the search area. This procedure is repeated until we have reached $R_1 = (I_1, \delta_1)$, in which case we use the rounding value of R_1 to define a disk proximity oracle and execute the `DiskSearch`$(\cdot)$ algorithm we presented. Specifically, we define the disk proximity oracle $P_{r,u}(\cdot)$ for a radius $r = \delta_1$ as follows:

$$P_{r,u}(p) = \begin{cases} 1 & \text{if } P_{S,u}(p) \leq \delta_1 \\ 0 & \text{Otherwise} \end{cases}$$

Notice that, after the last trilateration, the remaining points to search are located in an area of size δ_1^2. Therefore, a disk of radius δ_1 is large enough to run the `DiskSearch`$(\cdot)$ algorithm. Algorithm 1 shows the implementation of the attack. We define a procedure $B = \texttt{Trilaterate}(A, P_{S,u}(\cdot))$ which accepts as input an area A and a rounding proximity oracle, and by querying $P_{S,u}(\cdot)$ performs a trilateration to reduce the possible area, the user is residing in, to B. The details of the trilateration algorithm are known [22] and thus omitted; finally, the procedure `SimulateDiskProximityOracle` creates a disk proximity oracle as described before.

Analysis. The iterative application of trilateration is guaranteed to reduce the search area into a smaller rounding class every time, because we know that for a rounding class R_k we have that $\delta_k \leq \inf I_k$. Albeit natural, this restriction is not necessary; indeed, if at any point the algorithm cannot guarantee that trilateration will provide a reduced search area, then we can define the proximity oracle similarly to the way we defined it above and then execute the `DiskSearch`$(\cdot)$ algorithm. The number of queries the algorithm makes to the oracle is $3 \cdot |S| + O(\log \delta_1)$ where δ_1 is the rounding value of the smaller rounding class R_1. Additionally, the time required by the trilateration procedure is also constant and, thus, the total running time is $O(|S| + \delta_1 \log \delta_1)$.

Next, We prove two lemmas that establish a lower bound on the query complexity of any deterministic algorithm solving the rounding user discovery problem.

Lemma 5. *Let $P_{S,u}(\cdot)$ be a rounding proximity oracle indexed by a family of rounding classes $S = \{R_1, R_2, \ldots, R_l\}$ and an area A in the discrete Euclidean plane. Then, any deterministic algorithm solving the Rounding User Discovery Problem in the area A given access to $P_{S,u}(\cdot)$ requires $\Omega(\log \delta_1)$ queries to $P_{S,u}(\cdot)$ in the worst case.*

Proof. Let $S = \{(\mathbb{R}^+, \delta)\}$ and let A be a disk of radius $r = \delta/2 - \epsilon$, for some ϵ, $0 < \epsilon < \delta/4$. We will show that the rounding oracle $P_{S,u}(\cdot)$ can be simulated by an oracle which outputs only one bit of advice on each query. Afterwards, a straightforward application of the decision tree technique allows us to obtain a $\Omega(\log \delta)$ lower bound on the number of queries required to locate the user since the area contains at least $\Omega(\delta)$ points.

More formally, consider a query $P_{S,u}(p)$ on a point p. Denote by p_c the center point of the disk of the area A. Then, by the triangle inequality we have that

$$\text{dist}(p, p_c) - r \leq \text{dist}(p, p_u) \leq \text{dist}(p, p_c) + r$$

Notice that, within the interval $[\text{dist}(p, p_c) - r, \text{dist}(p, p_c) + r]$ there exists at most one integer m such that $\delta \mid m$. We define an oracle $O_u(p)$ as follows:

$$O_u(p) = \begin{cases} 0 & \text{if } \lceil \text{dist}(p, p_u) \rceil_\delta = \lceil \text{dist}(p, p_c) - r \rceil_\delta \\ 1 & \text{Otherwise} \end{cases}$$

Now the rounding oracle $P_{S,u}(\cdot)$ can be simulated as follows:

$$P_{S,u}(p) = \lceil \text{dist}(p, p_c) - r \rceil_\delta + O_u(p) \cdot \delta$$

[1] Since δ might not be an integer, we abuse the notation here to denote by $\delta \mid x$ that x is an integer multiple of δ.

Now we can use the same argument as in lemma 4: The binary decision tree based on the answers of the oracle $O_u(\cdot)$ must have $\Omega(\delta)$ leaves and therefore a height of at least $\Omega(\log \delta)$. □

Lemma 5 is interesting in the sense that it demonstrates that after the area is reduced into the order of the smallest rounding value, the rounding proximity oracle answers provide at most one bit of additional information in each query. Our next lemma states a lower bound on the queries required to reduce the distance to p_u into the interval of the smallest rounding class.

Lemma 6. *Let $P_{S,u}$ be a rounding proximity oracle indexed by a family of rounding classes $S = \{(I_1, \delta_1), \dots, (I_n, \delta_n)\}$ and an area A in the discrete Euclidean plane. Then, any algorithm for finding a point $p \in A$, such that $\operatorname{dist}(p, p_u) \in I_1$ requires at least $|S| - 1$ queries to $P_{S,u}$ in the worst case.*

Proof sketch. Let S be a rounding class family of size n such that for any rounding value $\delta_i, i > 1$, we have that $\delta_i \in I_{i-1} \wedge \delta_i > \pi \cdot (\inf I_{i-1} + \delta_{i-1})$. Moreover consider an area A where $|A| > \pi \cdot (\inf I_n + \delta_n)^2$. We will show that any problem instance in which the set S and the area A satisfy the above constraints requires at least $|S| - 1$ queries to $P_{S,u}$ in the worst case. The proof is by induction on the size of S.

- **Base case:** For $|S| = 2$, since $|A| > \pi \cdot (\inf I_2)^2$ at least one query is required in the worst case.
- **Inductive step:** Now, let us assume that for every set S of size at least k, any algorithm given as input an area A with size $|A| > \pi \cdot (\inf I_k + \delta_k)^2$ requires at least $k - 1$ queries to the proximity oracle $P_{S,u}$.

  Consider now a family S, with $|S| = k + 1$ and a single query of the algorithm at any point within an area of size $|A| \geq \pi \cdot (\inf I_{k+1} + \delta_{k+1})^2$. Then, notice that for any query of the algorithm on a point p, there exists at least one candidate point for the user such that $\operatorname{dist}(p_u, p) > \inf I_{k+1}$ and therefore, the candidate area for p_u is of size at least $\delta_{k+1}^2 > (\pi \cdot \inf I_k + \delta_k)^2 > \pi \cdot (\inf I_k + \delta_k)^2$. Therefore, the algorithm now has to search an area A' where $|A'| > \pi \cdot (\inf I_k + \delta_k)^2$ with the family of rounding classes $S_k = S \setminus \{(I_{k+1}, \delta_{k+1})\}$ and by the inductive hypothesis this requires at least $k - 1$ queries.

□

The following theorem follows easily from the two lemmas above, thus we omit its proof.

Theorem 3. *Any deterministic algorithm solving RUDP given access to a proximity oracle $P_{S,u}$ indexed by a family of rounding classes $S = ((I_1, \delta_1), \dots, (I_n, \delta_n))$, requires $\Omega(|S| + \log_{\delta_1})$ queries to $P_{S,u}$ in the worst case.*

2.3 Randomized User Discovery Problem

Under this model, the proximity oracle query result for a point follows a probability distribution rather than being a deterministic function of the distance. We capture this using the following definition:

Definition 9. *A randomized proximity oracle $P_{\mathcal{D},u}(p)$ indexed by a probability distribution $\mathcal{D}$ is defined as follows:*

$$P_{\mathcal{D},u}(p) \approx \mathcal{D}(p_u, p)$$

In other words, each point queried to the oracle will get its value from a probability distribution $\mathcal{D}$ based on the point queried and the location of the target user.

We define the randomized user discovery(RNDUDP) to be the UDP given access to a randomized proximity oracle. We also assume that the distribution $\mathcal{D}$ is known. Although it is unlikely in practice for a service to publish the distribution used, an attacker can employ a dummy victim user with a known location and collect a large number of samples from the distribution in order to create an estimation of $\mathcal{D}$.

Notice that the way RNDUDP is defined, there exist certain probability distributions for which it is impossible to locate the targeted user. For example, in the case where $\mathcal{D}$ is the uniform distribution over $\{0, 1\}$, querying the oracle will not provide any information on the location of the user.

Nevertheless, such distributions will also fail to offer any real functionality for the users of the service and therefore, they are not encountered in practice. Usually, the oracle $P_{\mathcal{D},u}$ will return the correct proximity information with some probability and an incorrect value otherwise.

For the following, we consider an intuitive randomization model which we denote as the **disk search with Gaussian noise**. Under this model the following randomized proximity oracle is defined.

Definition 10. *A randomized disk proximity oracle with radius r and standard deviation σ is defined as follows:*

$$P_{r,\sigma,u}(p) = (1 - z_{p_u}(p))P_{r,u}(p) + z_{p_u}(p)(1 - P_{r,u}(p))$$

where $P_{r,u}$ is a disk proximity oracle with radius r and $z_{p_u}(p) \in \{0, 1\}$ is the error function satisfying

$$\Pr[z_{p_u}(p) = 1 | dist(p, p_u) \in [a, b]] = \int_a^b f(x, r, \sigma) dx$$

where $a, b \in \mathbb{R}_+$ and $f(x, r, \sigma)$ is the PDF of $\mathcal{N}(r, \sigma)$, the normal distribution with mean r and standard deviation σ.

Intuitively, under this model, when an attacker attempts to detect points in distance r from the user in order to perform a cut, an increasing amount of noise is added forcing the attacker to perform a wrong cut with the DiskSearch algorithm and therefore stopping the attack. Moreover, the noise is reduced as we move away from that "danger zone", thus not affecting significantly the experience of normal users.

Out of all the different defenses we encountered in our research, this model was the most difficult to crack in the sense that it requires a much larger number of queries than the previous models. Below we present an algorithm in order to solve the DiskSearch problem with Gaussian noise, and also state a reduction showing that the problem of distinguishing between normal distributions with different mean values reduces to this problem.

Solving Disk Search with Gaussian Noise. Here we derive an algorithm to solve the DiskSearch problem with Gaussian noise. Our algorithm is an application of the Maximum Likelihood Estimation (MLE) principle which is widely applied in statistics and machine learning. Specifically, the algorithm proceeds as follows:

1. Given as input an area A, let r' be the radius of the minimum circle surrounding the area A. Then for a number k, given as input, sample k points randomly in distance $d \in [r - r', r + r']$ from the centroid of A. Let $S = \{(p_1, l_1 = P_{r,\sigma,u}(p_1)), \dots, (p_k, l_k = P_{r,\sigma,u}(p_k))\}$.

2. Compute the MLE over the candidate points in A as:

$$p_u = \underset{\hat{p_u} \in conv(A)}{\operatorname{argmax}} \sum_{i=1}^{k} \log \Pr[P_{r,\sigma,u}(p_i) = l_i]$$

where $conv(A)$ denotes the convex hull of the area A.

Analysis. For more information regarding the MLE algorithm we refer the reader to [21]. However, we note that the reason behind selecting the points within the convex hull of A is that since the probability density function of the Gaussian distribution is log-concave, thus the problem above can be written as a convex optimization problem and, if we select the points from within the convex hull of A, then one may utilize efficient convex optimization algorithms [13] in order to find the point with the maximum likelihood.

Moreover, we will show that solving the RNDUDP is at least as difficult as distinguishing between two gaussian distributions with different means and the same standard deviation. The problem of distinguishing between two normal distributions has been studied extensively in the past. We will show a simple lemma for the case of two univariate normal distributions with the same standard deviation and different means.

LEMMA 7. *Any algorithm distinguishing between the normal distributions $\mathcal{N}(r, \sigma)$ and $\mathcal{N}(r+\epsilon, \sigma)$ with constant probability requires $\Omega(\frac{\sigma^2}{\epsilon^2})$ samples.*

PROOF. It is known that distinguishing between two distributions P, Q requires at least $\Omega(1/KL(P, Q))$ samples [45], where KL is the Kullback-Leibler divergence [24] of the distributions. The KL divergence of two normal distributions is

$$KL(\mathcal{N}(\mu_1, \sigma_1), \mathcal{N}(\mu_2, \sigma_2)) = \log \frac{\sigma_1}{\sigma_2} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - 1/2$$

Replacing with the parameters of our distributions we get the result. □

We will show that that solving RNDUDP is at least as difficult as distinguishing between two normal distributions with different means and the same standard deviation. The two results together give a lower bound on the number of queries required to solve RNDUDP. In the following, accuracy denotes the distance between the point returned by the RNDUDP algorithm and the point p_u where the user resides.

THEOREM 4. *Let $P_{r,\sigma,u}$ be a randomized proximity oracle instantiated with a normal distribution $\mathcal{N}(r, \sigma)$. Then, any algorithm solving RNDUDP with constant probability and accuracy $\epsilon \ll r$, requires $\Omega(\frac{\sigma^2}{\epsilon^2})$ queries to the randomized proximity oracle $P_{r,\sigma,u}$.*

PROOF SKETCH. We consider an easier case of *RNDUDP* where there are only two candidate points for the targeted user in distance ϵ. Without loss of generality assume that the candidate points are $p_1 = (0, 0)$ and $p_2 = (0, \epsilon)$. The attacker has therefore to choose between the targeted user being in point p_1 or point p_2. Notice now, that in case the attacker makes all his queries on the line defined by points p_1 and p_2, then the problem reduces to an instance of distinguising between the distributions $\mathcal{N}(r, \sigma)$ and $\mathcal{N}(r + \epsilon, \sigma)$. Moreover, any query performed in a different angle is suboptimal; indeed, let p be any point in the plane. Then by the triangle inequality we have that

$$dist(p, p_1) - dist(p, p_2) \leq \epsilon$$

with equality holding only in points lying on the line defined by p_1, p_2. Thus, under different angles the attacker will have to distinguish between normal distributions with a mean that differ less than ϵ which, by lemma 7, is harder than distinguishing two distributions with mean difference ϵ. Thus, the best strategy for the attacker is to make all the queries in the line defined by p_1, p_2 essentially reducing to the problem of distinguishing between $\mathcal{N}(r, \sigma)$ and $\mathcal{N}(r + \epsilon, \sigma)$.

□

3. PRACTICAL DIMENSIONS OF UDP

Here we describe the practical aspects and challenges of carrying out the attacks against actual services.

Search space. Depending on the adversary's knowledge, the search space may be significantly large. Chaabane et al. [14] found that 29% of users publicly disclose their current city, while 48% disclose it to their contacts. Such information can, potentially, also be obtained through a geo-inference attack [23]. While our attack is very efficient, collateral information can reduce the size of the search space.

Proximity Oracle Querying. As the attack requires interaction with a service, the following become relevant.

Connections. Services may restrict which users can obtain proximity information. Dating apps that offer a platform for meeting new people follow a less restricted approach and disclose that information to users that are not connected (i.e., strangers). Social networks have more privacy-oriented default settings, and disclose the information to the users' contacts. While this might seem like an obstacle, the adversary can employ fake accounts to befriend the user. Previous work (e.g., [12]) has explored this topic extensively, and demonstrated that users are very likely to accept requests from unknown accounts. For the remainder of the paper we assume that, if required, the adversary is connected to the user within the service. Nonetheless, we minimize the number of attackers; when a social connection is required we employ a single account.

Detection. As it is trivial to send fake coordinates, services may deploy mechanisms for detecting this. To identify the mechanisms and their thresholds, we follow the approach from [36]. By knowing the exact thresholds, we minimize the duration of the attack while avoiding detection.

Attack Accuracy. The accuracy is unavoidably dependent on the GPS precision of the user's device. In our experiments, distances are calculated from the coordinates reported to the service. Thus, we measure the *exact accuracy* of our attacks. In practice, the accuracy may be influenced by the error of the device's GPS. However, smartphones are becoming increasingly accurate, with GPS accuracy being within 10m and researchers demonstrating improvements of 30% [44]. Furthermore, the accuracy of our attack is such that GPS errors do not alleviate the threat to targeted users.

Projection Errors. No map projection perfectly represents all areas, and a given projection is constructed to preserve only a few properties (e.g., shape) of the real geographical area. While the attacks are not dependent on the projected coordinates, the coordinates of the search area should be transformed using the projection most suitable for that area, to minimize the error introduced. We use the appropriate equidistant conic projection for the continent we are searching in, centered at our area of interest.

	#	Dst	GPS	Grid	Query	Speed	Rand
Facebook	1-5B	◎	✗	✗	✗	✓	✗
Swarm	5-10M	◯	✓	✗	✗	✓	✓
Grindr	5-10M	⊙ | ∅	✗	✗	✓	✗	✗
Skout	10-50M	◎	✗	✗	✓	✓	✓
MeetMe	10-50M	◯ | ◎	✗	✓	✗	✓	✗
Lovoo	10-50M	◎	✗	✗	✗	✗	✗
Tinder	10-50M	◎	✗	✗	✗	✓	✗
SayHi	10-50M	⊙	✓	✗	✗	✗	✗
Jaumo	5-10M	◎	✓	✗	✗	✗	✗
HelloWorld	1K-5K	◎	✗	✗	✗	✗	✗

Distance: exact ⊙ | rings ◎ | disks ◯ | none ∅

Table 2: Popular services with location proximity.

4. LOCATION PROXIMITY: A CASE STUDY

Here, we provide an overview of our audit of popular apps that offer location proximity. We analyse several aspects of the apps, to verify the applicability of our attacks. Table 2 presents their description, and the number of downloads (#) reported by Google's Play Store. Upon auditing these apps, we found that our theoretical formulation of the discovery problem and the proximity oracle constructions are complete, and can model the proximity functionality of existing services. For each app, we explore the following:

Distance information. The services may reveal the distance in various forms (e.g., disks, rings etc.).

GPS coordinates. The user's GPS coordinates may be shared with other users; this can be done explicitly (e.g., exact location shown) or implicitly (app receives coordinates of contacts for calculating distance).

Grid. The service may use a grid when calculating distances. In Swarm, while a grid is employed, it does not affect the calculation. In MeetMe, the distance is calculated from the adversary's grid point to the victim's actual location. Thus, neither perform spatial cloaking.

Query limiting. The service may enforce query rate limiting that prohibits a straightforward execution of the attack. Unless a very strict limit is enforced, one can simply increase the waiting time between queries.

Speed constraints. The service may enforce a limit on the speed with which users travel between locations.

Rand. The service adds random noise in the distances returned by the proximity oracle.

Next, we provide details for the selection of apps that we evaluate in Section 5. This set of apps allows us to examine our proximity oracle construction against all forms of distance disclosure. Furthermore, they have adopted a diverse set of countermeasures designed to prevent user discovery attacks and, thus, pose interesting challenges.

Swarm was deployed by Foursquare and is built around location proximity. Friends are assigned to location disks of six sizes: (i) 300 m, (ii) 1.5 km, (iii) 15 km, (iv) 30 km, (v) 64 km, and (vi) everything over 64 km. The API used by Swarm has not been made public. As such, we reverse engineer the communication protocol between the app and servers. Our experiments reveal that Swarm employs a static grid, with users being assigned to their nearest grid point. The grid cells' dimensions are $252m * 274m$. There are two relevant API calls; `UpdateLocation` updates the location and is periodically sent by the app. `Activities_Recent` is sent upon interaction with the app and contains the user's current location. The response contains the active contacts' grid point coordinates and their distance to the sent location. The sent location is not saved, but only used to calculate the distance to the contacts. We use this API call to place our attacker at arbitrary positions. Swarm limits the number of API requests to 500 per hour. Furthermore, speed constraints are only enforced for `UpdateLocation` (we bypass them by using `Activities_Recent`).

Facebook. Two private Graph API calls implement the required functionality; updating the current location and querying for friends in proximity. Our experiments reveal that the coordinates sent with the `nearby-friends` call, are not correlated to those received by the `update-location` call which is considered the user's last known location. While Facebook enforces speed constraints, attackers can exploit this discrepancy to bypass them. Thus, the adversary can carry out the user discovery attack and query the service from multiple vantage points without any speed restriction.

Grindr. To prevent user discovery attacks [2], the new default setting for users located in countries with oppressive regimes is to completely hide the distance information. For other users, exact distances are disclosed. However, the app displays nearby users sorted by distance, regardless of their settings. While this prevents trilateration attacks, Grindr remains vulnerable to our attack. As Grindr is intended for meeting new people, no social connection is required.

Skout. We reverse engineer the app, and identify two API calls that use `SOAP` request and response headers. One sets a user's location, while the other takes as input a `userID` and retrieves the user's information (including the distance).

5. EXPERIMENTAL EVALUATION

Here we evaluate our attacks against actual services. We deploy virtual users that play the role of the target. As our attack system sets the users' exact location (*ground truth*) in the service and uses it to calculate the distance, our evaluation presents the *exact accuracy* of our attack and does not contain any measurement error (e.g., due to GPS). Since Foursquare and Facebook require a social connection, we demonstrate our attacks using *a single attacking account.* As no connection is required in Grindr and Skout, the attacks can use multiple accounts in parallel to reduce the duration; thus, we do not report an attack duration for those services.

5.1 Foursquare - Swarm

Our experiments show that location updates are disseminated in under a second, enabling real time user tracking.

Projection Error. To measure the impact of our projection error, we run 100 attacks in each USA state, with Wally placed randomly within the state. The response to our API call contains the coordinates of the grid point that Wally has been mapped to, and a distance to Wally. Here we consider Wally's location to be the grid point and discard the distance information. Using the coordinates of the grid point we simulate a disk proximity oracle: we measure the distance from Mallory's location and select the corresponding disk size, as shown in the Swarm app (see Section 4). Subsequently, we run our DUDP and `DiskSearch` algorithms to identify Wally's location. In the majority of states the projection error is negligible and we achieve distances of less than 1 meter, by centering the projection over the state. For 15 states the distance increases to less than $5m$, and for 5 states to less than $10m$. These results are sufficiently precise for evaluating our attack. We run our experiments

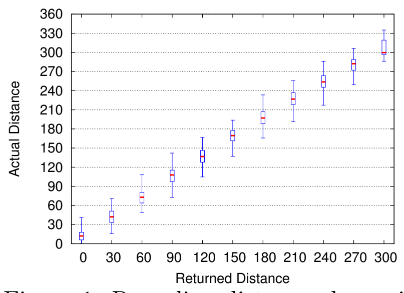

Figure 1: Rounding distance classes in Swarm.

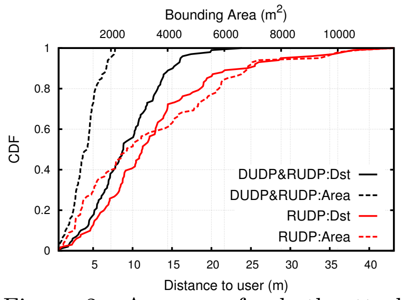

Figure 2: Accuracy for both attacks against static user in Swarm.

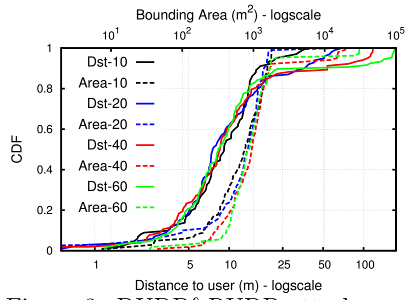

Figure 3: DUDP&RUDP attack accuracy for moving user in Swarm.

in New York where the error is less than 1 meter.

Complete attack. Next, we evaluate the accuracy of our attack in identifying Wally's exact location and not the grid point he has been mapped to. As such, we also use the distance information returned. Our experiments reveal that the returned distances are calculated based on Wally's actual location and not the grid point. As can be seen in Figure 1, Swarm returns quantized distances in rounding classes of $30m$. Furthermore, an amount of randomness, which depends on the victim's position within the grid cell, affects the rounding process as an extra measure of privacy. Using our empirical measurements, we translate the returned distance into a location ring. We follow two attack methodologies. In one case, we run the Disk User Discovery, Rounding User Discovery, and `DiskSearch` algorithms and achieve maximum accuracy (with an overhead in terms of queries). In the other, we only employ Rounding User Discovery, and `DiskSearch`. While the accuracy slightly deteriorates, the attack is more efficient in terms of queries and can be used in services with strict query limiting or speed constraints.

DUDP&RUDP attack. Figure 2 presents 100 experiments carried out in the state of New York using a combination of the DUDP and RUDP algorithms. Apart from accurate, the attack is also efficient, with an average of 55.9 queries and 6.87 seconds required for each attack. Half the users are pinpointed within 9 meters, and 95% within 16 meters. When it comes to the bounding area in which the targeted victim can be found, in 20% of the cases it is less than $700m^2$. For half the attacks it is up to 1,160 m^2, and up to 2,000 m^2 for 95%. Thus, for 95% of the users the attack reduces the bounding area by at least 99.29%, compared to the smallest disk disclosed by the service. If Mallory's goal is to bound the user within a specific area for inferring personal data (e.g., a clinic close to the returned location), she can probabilistically bound the user to an arbitrarily small disk, based on these empirical measurements. For example, there is a 50% chance of of the user being located within a bounding area of $255m^2$ (a disk with a $9m$ radius). Depending on the attack scenario, the adversary can select a different area size, based on the area/probability trade-off.

RUDP attack. Figure 2 also presents 100 runs of the resource-efficient attack, which employs the RUDP algorithm; 40% of the users are found within 10m, and 95% within 30m. The bounding area is also less accurate, with 22% up to $1,000m^2$, and 50% less than $2,600m^2$. Thus, for half of the users, the bounding area is reduced by 99%. While less accurate, this approach is far more efficient with an average of 18.2 queries and a duration of 2.59 seconds.

Maximum Area. The disk coverage algorithm can cover the entire continental USA with 850 queries, when using disks with $r \simeq 64.3km$ (maximum supported by Swarm). Considering the 500 query limit, one can cover a $\sim 6,1 \times 10^6$ km^2 area with 475 queries prior to launching `DiskSearch`. Thus, the attack is very practical even at a continent-level. At a state-level, the area can be covered with significantly less queries (e.g., at most, 98 for Texas, 28 for New York).

Moving Targets. We explore if our attacks can track a moving user in real time, and test speeds ranging from that of a human running to a vehicle moving in a city. Both Swarm and Facebook update the location every several minutes, and upon interaction with the app. Thus, updates are too infrequent to impact the attack, and our accuracy remains the same. Thus, we run an experiment to "stress test" our attack, with users moving at constant speeds and their location updated every 10 seconds. Then, at random intervals, we run our attack against those users, and calculate the distance between the inferred and exact locations at that moment in time. While not a realistic setup, as in reality targets are not in continuous motion but stop intermittently (e.g., traffic lights), it poses a worst-case scenario and demonstrates how users can be tracked even when in motion. In practice, the accuracy will be even higher.

For the moving victim, we modify the algorithm to provision for the changing location of the victim. Since the user is moving, even if he is bound within a particular area P_i at a certain time t_i, it does not necessarily hold that he will also lie within the same area in a future point at time $t_j > t_i$. If at time t_j we query the oracle for the location of the victim and the response is an annulus A_j, normally the new polygon in which the victim is bound is $P_j = A_j \cap P_i$. For a moving victim though, it may hold that $A_j \cap P_i = \emptyset$. In those cases we set $P_j = A_j \cup P_i$. *We do not utilize any information regarding the user's speed.*

Figures 3 and 4 present the results for the accurate (DUDP & RUDP) and efficient (RUDP) attacks respectively, for users moving at speeds of $\{10, 20, 40, 60\}$ km/h. For targets simulating a user running (10kmh), our accurate attack is almost the same as for static users, with 90% located within 16.5 meters, while the efficient one locates 67% within 20m. As expected, the attack's accuracy deteriorates as the speed increases. Nonetheless, even when the user maintains a steady speed of 60kmh (average speed for US cities is 28-70 kmh [1]), the accurate attack achieves a distance less than 10m away for 60% of the experiments, and 85% is up to 20m. The efficient attack pinpoints 60% of the victims within 20m. For users moving at $60kmh$, ten experiments returned a location more than 75m away, due to the location being updated during the attack. Depending on the update frequency, the adversary can provision against this by running successive attacks. Since the goal is to track the

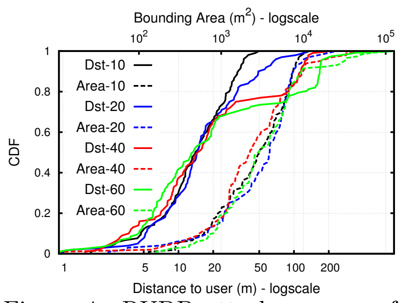

Figure 4: RUDP attack accuracy for moving user in Swarm.

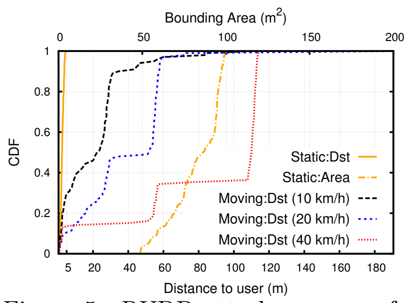

Figure 5: RUDP attack accuracy for static and moving user in Facebook.

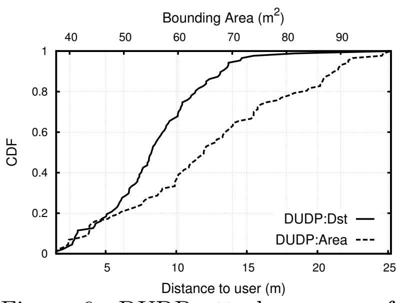

Figure 6: DUDP attack accuracy for static user in Grindr.

user while moving, and not to identify a specific location, the bounding areas do not affect the objective of the attack.

Overall evaluation. By disclosing the (rounded) distance from the user's actual position, the privacy that could be guaranteed by the grid is lost. The randomness introduced is insufficient for preventing our attack, which eliminates the noise without using the RANDUDP algorithm.

Demonstration. We have a front-end for visualizing our attacks[2]. Our back-end carries out an attack in real time, and the front-end visualizes it at a lower speed.

5.2 Facebook

Facebook rounds the distance information (0.5 mile granularity for distances less than 1 mile, and 1 mile granularity for distances up to 100 miles). Thus, we employ the RUDP attack algorithm. We evaluate the effectiveness of our attack by carrying out experiments for static and moving victims. For each experiment, we place the victims randomly in 100 locations in New York. Each attack requires about 20.5 queries and is completed within 3.05 sec. As shown in Figure 5, when targeting a static user, *the estimated locations are always less than* 5*m away from the user's exact location*, and the victim in bound within an area less than $100m^2$. For moving victims, the quality of the attack is significantly reduced for larger speeds: for targets moving at 40km/h, 40% of the users are less than 110m from the estimated location. Due to the RUDP attack lasting slightly longer than for Swarm, there is an increased chance of the location being updated during the attack, which results in the decreased accuracy. Nevertheless, adversaries can still track a user in motion. Overall, our results demonstrate that distance rounding is ineffective at preserving privacy, as static users are pinpointed within 5m of their exact location.

5.3 Grindr

Due to the sorting of users, if Mallory knows the location of another user she can deduce information about the distance to Wally, by comparing that user's index (position within the user list) to Wally's. Mallory can leverage this information for defining a *disk proximity oracle* with a radius of her choice. The attack is then reduced to an instance of the disk user discovery problem. Mallory creates two accounts, and issues a disk proximity oracle query $P_{r,u}(p)$ for a point p as follows. The attacker is placed at the point p, and the colluder is placed at a distance r from p. The attacker pulls nearby users from the service. If the target has a smaller index than the colluder, the oracle replies to the query with 1, otherwise with 0.

[2]A demo can be found at: http://youtu.be/7jCr36IT_2s

Query responses contain the 50 nearest users in ascending order. Due to the large number of users within an entire city, which results in an increased number of requests towards the service for fetching the pages with users from each probing point, we opt for a modified attack setup. We run experiments where Mallory knows that Wally is in a certain area of New York; we set that area to be a disk with a 600m radius, resulting in a search space of $(1.13 * 10^6)m^2$. The results from our experiments are shown in Figure 6. Again, the attacks are very effective with 67% of the users located within 10m, 98% within 19m, and in one case 25m away. All users are bound within a $100m^2$ area, while the attack requires a total of 58 queries on average. This includes requesting extra pages when Wally is not found within the 50 first results. An adversary can deploy real time attacks in much larger areas, by employing multiple accounts.

This case highlights an important detail; if Mallory can discover even a bit of information for inferring the user's proximity to a point in the plane, she is able to precisely determine the location. This is useful for assessing the privacy of models that conceal the precise distance, but still leak some side channel information about it.

5.4 Skout

Skout implements a randomized proximity oracle for defending against user discovery attacks. Previous work [25] reported that Skout only used quantized distances. Our experiments reveal that the current version has a far more advanced defense mechanism, which adds Gaussian noise to the reported distances as described in section 2.3. To evaluate the efficiency of the implemented defense, we deploy our RANDUDP attack. We stress that sampling many points at a close distance for estimating the error probability in a specific area does not work, because points in Skout are labeled in clusters, with nearby points getting the same label (either correct or incorrect) with very high probability.

In order to estimate the error probability distribution we employed two accounts and measured the probability of getting an erroneous distance in different locations and distances. In Figure 7 we show the error distribution as a function of the distance and the resulting Gaussian distribution obtained from the samples. In order to fit the sampled distribution in a Gaussian distribution we used the Levenberg-Marquardt algorithm [29], from which we obtained a Gaussian with mean $\mu = 0.5$ miles and standard deviation $\sigma = 0.039$ miles.

Even though Skout uses rounding classes (all distances are rounded up using the $\lceil \cdot \rceil_{0.5}$ operator) by using a RUDP attack we reduce the candidate area A to an area which can be enclosed in a circle of radius $r' = 0.1$ miles. Therefore, the

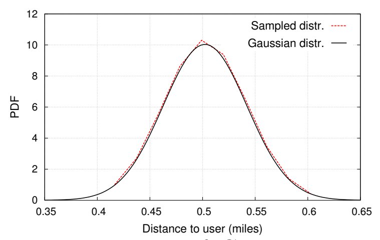

Figure 7: PDF of Gaussian error distribution in Skout.

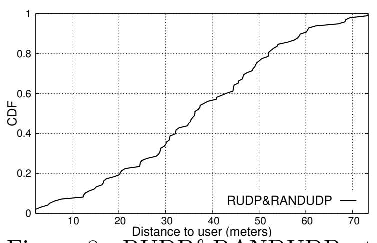

Figure 8: RUDP&RANDUDP attack accuracy for Skout.

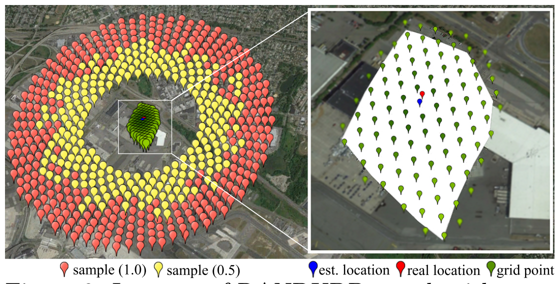

Figure 9: Instance of RANDUDP attack with sampling points for the MLE calculation.

problem effectively reduces to an instance of the disk search problem with Gaussian noise. Afterwards, we execute the algorithm for solving the disk search problem with Gaussian noise, with the parameters described above.

In Figure 9 we use a red pin to depict the points for which Skout returned a distance of 1.0 and a yellow pin to depict those reported as being 0.5 miles away. We notice that, even though the points that are at most 0.3 miles are consistently rounded up to 0.5, as Mallory enters the range between 0.4 and 0.6 miles, the returned distances approximate the Gaussian distribution. The green pins represent candidate points for the location of the target, with darker colors denoting higher likelihood. Wally's estimated location is shown with a blue pin and the real location with the red pin.

To evaluate our approach, we perform 100 attacks with the user placed randomly in New York. For the MLE calculation, we use 600 sample points. Each attack requires an average of 1,400 queries, and we use multiple accounts for efficiency. The attack's accuracy is shown in Figure 8; on average we pinpoint Wally's location within 37.4m despite the advanced defense mechanism employed. Our attack achieves substantially better results than previous attempts [25], even though Skout now implements a far more sophisticated defense. While this attack requires a much larger number of queries, Skout does not require a social connection for obtaining proximity information. Thus, while this defense would significantly raise the bar in a service that requires a connection (e.g., Facebook), its effect in Skout is diminished, as creating multiple accounts is not a major obstacle.

6. PREVENTING DISCOVERY ATTACKS

Based on our theoretical models and experimental evaluation, we explore the use of *spatial cloaking* for ensuring a minimum level of privacy in proximity services. We set the following requirements for the service:

- The system should ensure that the user cannot be bound to an area smaller than a fixed region.
- Users' coordinates should never be explicitly disclosed to other users (or sent to the app on their device).

Spatial Cloacking. Employing grids for creating cloaking regions has been proposed in the past. We revisit the concept and analyze its properties based on the characteristics of our attacks. The solution we present is similar to the proximity protocols presented in [41], in the sense that it works by rounding the coordinates of the user's location, effectively creating a grid over the Euclidean plane. This can be easily adopted by any proximity service.

Grid construction. Assume that a user is located at a point $p = (x, y)$ in the Euclidean plane. The service uses a predefined rounding value δ and maps the user to the location $p_c = (\lfloor x \rceil_\delta, \lfloor y \rceil_\delta)$. This rounding operation maps all locations within a square grid cell of edge length δ to the centroid of that grid cell. The main questions that we would like to answer for our construction are:

1. How much *privacy* does a grid cell of size δ offer?
2. Is *usability* maintained at an acceptable level?

Privacy. We first notice that the bounding area for a static user inside a grid cell cannot be better than δ^2. The attacker however, wants to minimize the distance from the target user. Let's first assume that an attacker runs our attack and finds the grid cell of the user; lets also assume that the user is randomly placed at a point p_u within the cell. To minimize her distance from the targeted user she will place herself at the centroid p_c of the grid cell. Since the accuracy of an attack depends on the distance between the attacker and the user, we would like to estimate the expected value of $\text{dist}(p_u, p_c)$. In appendix A.1, we prove the following:

$$\mathbb{E}[\text{dist}(p_c, p_u)] \approx 0.382598 \cdot \delta$$

For example, using a grid size of 0.5 miles the attacker will be able to find the user within 0.19 miles on average. The *expected distance* value is an appropriate metric for evaluating the privacy offered by a grid cell of edge length δ in the context of user discovery attacks. As the user won't always be at a random location within a grid cell, social information can be used to further refine his location after the grid cell is detected. Our metric quantifies the privacy obtained by a grid cell of a certain size, when the adversary lacks knowledge of a user's past movements.

For proximity protocols where the sender can select the receiver's grid size ([34, 47]), if a safe minimum size is not enforced, an adversary can deploy our attacks and bound the user within a small cell that fails to offer any privacy. As our attack algorithms are designed for a discrete plane, they can handle grids without modifications.

Usability. Another important parameter is quantifying how much this solution will affect the usability of the service. Let us assume that two users u_1, u_2 are mapped to different grid cells p_1, p_2 with a grid edge of length δ. Computing the distances from the rounded coordinates incurs only a small error in the computation. With a simple application of the triangle inequality we have that:

$$\text{dist}(p_1, p_2) \geq \text{dist}(\lfloor p_1 \rceil_\delta, \lfloor p_2 \rceil_\delta) - \sqrt{2}\delta$$
$$\text{dist}(p_1, p_2) \leq \text{dist}(\lfloor p_1 \rceil_\delta, \lfloor p_2 \rceil_\delta) + \sqrt{2}\delta$$

$\text{Dist}(p_1, p_2)$ is at most $\sqrt{2}\delta$ off from the actual distance.

Tradeoff. We discuss the performance of our solution compared to current implementations, where rounding is applied to calculations, in terms of usability and privacy. As an example, we consider Facebook: the minimum rounding value applied is 0.5 miles. Consider now a grid cell of length 0.5

miles: as shown, a grid size of 0.5 miles will provide the user with an average privacy of 0.19 miles from attacks, and the distance provided to other users will be off by at most $\sqrt{2} \cdot 0.5 \approx 0.7$ miles. Thus, compared to the current implementation, the grid will add an additional error of at most 0.2 miles to the distance calculations, while providing a much better privacy guarantee. For distances over 1 mile, which are rounded up by 1, this does not incur any additional error to the rounding already added by the service.

Protecting Moving Users. While a grid ensures privacy down to a certain granularity for static users, maintaining the same level of privacy for moving users becomes complicated. By polling for a user's location, the adversary can infer a user's speed and direction of movement. Based on that knowledge, and identifying when the user crosses the boundaries of a cell, the user can be bound within an area smaller than the cell. A large body of work has explored methods for protecting [18, 39, 40, 42] and attacking [20] moving users. The solutions vary based on the information available to the attacker and the threat model. Most share a common assumption: the existence of an obfuscation mechanism which, given the user's precise location, generates a pseudo-location. Thus, our grid proposal is *orthogonal* to the privacy schemes for protecting moving users, as it provides the necessary foundation for deploying them. In addition, the formal framework by Shokri et al. [39], supports our grid construction as an obfuscation mechanism, and can quantify the location privacy of moving users, given a specific Location Privacy Protection Mechanism.

Location Verification. A crucial aspect of the attack is the ability to arbitrarily change positions when querying the oracle. Implementing a mechanism for verifying the user's location could significantly impact the attack.

Mitigation. We discuss measures that are trivial for the services to implement and can impact the attacks' performance without degrading usability, until a grid is deployed.

Query rate limiting. Adversaries should be prohibited from flooding the service with queries for a specific user. Even if the oracle can only be queried once every 5 minutes for a user, the attack would require significantly longer time (90 minutes for the efficient attack in Swarm, 100 minutes in Facebook). Nonetheless, adversaries will still be able to identify locations where the user remains for long periods of time. If no connection is required, the adversary can use multiple accounts, to carry out the attack faster. *The cost of creating accounts should not be considered prohibitive.*

User movement constraints. Services may employ detection mechanisms for identifying users that change locations unrealistically fast. To avoid detection the adversary will have to remain beneath a speed threshold when moving between probing points, which will increase the attack's duration. However, while the attack's duration will increase, for these measures to be able to significantly impact the attack, the service would have to impose strict constraints that would also impede legitimate users. Furthermore, adversaries can identify the thresholds and configure their attack to maximize the efficiency while remaining undetected [36].

Distance disclosure. If large distances are disclosed by the service (e.g., $10km$), the adversary can employ the disk coverage algorithm and greatly reduce the search space with a few queries. Thus, the service should only disclose small distances, which are large enough to retain usability.

Auditing Framework. We have developed a framework to facilitate researchers, developers, and privacy-sensitive individuals, in assessing proximity services. This can lead to the evaluation of a large collection of apps, and help improve the privacy offered by existing services. It has been implemented in Python and released as an open source project[3]. The framework provides tests for automatically evaluating services with disk or rounding proximity oracles against our attacks. We are currently expanding it to also handle randomized proximity oracles. It also provides a set of libraries that facilitate common operations on geographical coordinates (natural and projected). Since we cannot predict the requirements of each app, auditors have to implement the app-specific calls for (i) setting a user's location and (ii) getting the distance to a user. Once the auditor implements the above calls, both the constraints-testing and the attacks work out of the box. If a service is not a straightforward application of the RUDP or DUDP proximity oracles (e.g., as is Grindr, where the oracle is constructed based on the ordering of users), auditors can specify their own proximity oracle, which can be passed as a parameter to our framework for automatically evaluating the service.

7. ETHICS AND DISCLOSURE

Disclosing attacks against services raises ethical issues as, one might argue, adversaries may have previously lacked the know-how to conduct such attacks. However, recent events demonstrated that adversaries are already targeting users. While the attacks seen in the wild were straightforward, and preventable by existing defenses, it is crucial to assess the privacy guarantees of popular services and prevent future atttacks. The false sense of security cultivated by the services and major media, due to misconceptions of the privacy offered [4, 5] and flawed recommendations (e.g., rounding distances [3]), exposes users to significant threat. We contacted the services we evaluated, and provided them with a detailed analysis of our attacks, our guidelines for preventing them, and an analysis on the privacy/usability tradeoff of our spatial cloaking approach. As a result of our disclosure, Foursquare and Facebook have adopted spatial cloaking. Furthermore, the Facebook security team used our testing framework for evaluating their cloaking mechanism. Grindr and Skout have not yet informed us of changes.

8. LIMITATIONS AND FUTURE WORK

Past locations. When previous locations of the user are known, the adversary can potentially reduce the search space of the attack. Even if the service uses spatial cloaking, the adversary may probabilistically map the user to certain locations within the grid cell; i.e., if the user is known to frequent locations X and Y, and the adversary traces the user to a grid cell which contains X and Y, then it is likely that the user may be at one of those locations. Temporal information may further increase the probability (e.g., the user visits X every Friday night). Allowing the user to dynamically configure the size of grid cells as certain sensitive areas, could increase the number of potential locations within a cell and decrease the certainty of the attacker for each location. This, however, remains out of scope of this paper, and is considered future work.

[3]Code available at: http://www.cs.columbia.edu/nsl/projects/vpsn/

Client side. Our proposed solution can be adopted independently by users and retain the same guarantees for usability and privacy, as if it were constructed by the service. Similarly to apps that spoof GPS coordinates, an app can generate a quantized version of the coordinates, effectively supplying services with a cloaked grid point instead of the real location. In rooted Android environments where more control is given over individual permissions, we can enforce different grid sizes for each service. We plan to develop such an app to assist users in maintaining their privacy, even if services fail to correctly obscure their location.

9. RELATED WORK

Location Proximity. Zhong et al. [47] presented three private proximity protocols. Similarly, Narayanan et al. [34] proposed multiple protocols for private proximity testing where untrusted servers mediate communication. However, in their system the sender configures the grid's size, which allows adversaries to use our attack technique and bind the user within a cell which is too small to offer any privacy.

Mascetti et al. [31] proposed privacy-preserving techniques and explored the trade-off between achievable privacy and quality of service. In [30] they presented an attack against users that obfuscate their location with dummy queries; it uses clustering, population density data and trilateration, to bound users within an area. The attack only works for users located in large cities and under unrealistic assumptions of uniform user density. Even then, the bounding area is too large to locate users or infer sensitive data.

Puttaswamy and Zhao [37] proposed moving application functionality to client devices, and treat services as untrusted storage for encrypted data: thus, sensitive information in never disclosed to untrusted parties. Šikšnys et al. [41] presented FriendLocator, which employs multiple levels of grids of varying size to calculate proximity while hiding the location from the service. Our attack can be carried out, starting at the level with the largest cells and recursively descending levels until the smallest cell is reached. If the smallest cell is sufficiently large a minimum level of privacy can be ensured.

Location Verification. Narayanan et al. [34] proposed location tags for proximity verification protocols. Lin et al. [26], explored the use of GSM cellular networks for creating location tags. While a promising approach, it presents significant practical limitations, as users must be connected to the same base station tower. Marforio et al. [28] proposed the use of smartphones in a location verification scheme for sales transactions. A similar approach could potentially be employed by proximity services for verifying users' location.

Location Privacy. Gruteser and Grunwald [19] introduced spatio-temporal cloaking for providing coarse-grained information to a service, while retaining functionality. Shokri et al. [39] provided a formal framework for quantifying the privacy guarantees of location privacy protection mechanisms, and a model for formulating location information disclosure attacks. In follow up work [40], they presented a framework that enables a service designer to find the optimal privacy preservation mechanism tailored to each user's desired level of privacy and quality of service. Their model is designed for passive adversaries (eavesdroppers), while we focus on active adversaries that interact with the service. Andres et. al [11] employ randomization to protect the location privacy of users through the notion of geo-indistinguishability. Nevertheless, the level of privacy provided by their mechanism degrades linearly with the number of queries to the LBS. A comparison to RANDUDP is an interesting future research direction. Fawaz and Shin [17] present a client side framework for protecting the location privacy of users that employs the approach of [11].

Recent work has demonstrated straightforward trilateration attacks against services that return exact distances [38]. In an independent recent study, Li et al. [25] explored user discovery attacks and highlighted the significance of this threat. The main limitation of that work is the inability to effectively account for the noise introduced by these services as a defensive measure, which manifests as location inaccuracy and leads to tracking results that are too coarse for most urban settings and many other interesting scenarios (e.g., identifying the user's home or workplace). Furthermore, the substantial duration of their attacks (6.5 to 30 minutes) is prohibitive for realistic attack scenarios. On the contrary, we follow a systematic approach and formulate the attacks as algorithmic problems, which enables the design of optimal attacks against all types of proximity services that we have encountered. We extensively explore the details and challenges of deploying such attacks in practice, and demonstrate the effectiveness and efficiency of our attacks against a wide range of defenses.

10. CONCLUSIONS

In this paper, we explored the privacy guarantees of location proximity services. We formalized the problem of user discovery attacks, and designed algorithms with query complexities that match the proven lower bound within a constant factor. We systematically evaluated four popular services and identified the practical challenges of deploying user discovery attacks in the wild. Even though services have deployed countermeasures to prevent trilateration, our experiments demonstrated that they remain vulnerable to more advanced attacks. Based on the insight obtained from the experiments, and the characteristics of our attacks, we proposed a set of guidelines for implementing spatial cloaking in proximity services, which ensures a minimum level of privacy. Disclosure of our findings and guidelines led to the adoption of spatial cloaking by major social networks. Thus, we have made a concrete contribution to user privacy, which resulted in the protection of users against the very effective and efficient class of attacks we have demonstrated. However, many popular apps remain vulnerable, exposing their users to significant threats. As such, we have released an auditing framework for assisting developers and researchers in evaluating the privacy offered by proximity services.

11. REFERENCES

[1] CitySpeed - Road network efficiency.
[2] Grindr - Location Security Update. (09/2014).
[3] Synack Security - The Do's and Don'ts of Location Aware Apps; A Case Study. (09/2014).
[4] Techcrunch - Facebook Nearby Friends. (04/2014).
[5] Time - Tinder Security Flaw Exposed Users' Locations. (02/2014).
[6] Huffington Post - Feds Using Fake Online Profiles To Spy On Suspects. (03/2010).
[7] The Independent - Grindr and Egypt. (09/2014).
[8] Police Forum - Social Media and Tactical Considerations For Law Enforcement. (2013).

[9] Washington Post - Could using gay dating app Grindr get you arrested in Egypt? (09/2014).

[10] WSJ- Stalkers Exploit Cellphone GPS. (08/2010).

[11] Andrés, M. E., Bordenabe, N. E., Chatzikokolakis, K., and Palamidessi, C. Geo-indistinguishability: Differential privacy for location-based systems. In *ACM CCS '13*.

[12] Bilge, L., Strufe, T., Balzarotti, D., and Kirda, E. All your contacts are belong to us: automated identity theft attacks on social networks. In *ACM WWW 2009*.

[13] Boyd, S., and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004.

[14] Chaabane, A., Acs, G., and Kaafar, M. You are what you like! information leakage through users' interests. In *NDSS '12*.

[15] Clark, B. N., Colbourn, C. J., and Johnson, D. S. Unit disk graphs. *Discrete Mathematics* (1990).

[16] De, M., Das, G. K., and Nandy, S. C. Approximation algorithms for the discrete piercing set problem for unit disks. In *CCCG* (2011).

[17] Fawaz, K., and Shin, K. G. Location privacy protection for smartphone users. CCS '14'.

[18] Ghinita, G., Damiani, M. L., Silvestri, C., and Bertino, E. Preventing velocity-based linkage attacks in location-aware applications. In *GIS '09'*.

[19] Gruteser, M., and Grunwald, D. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys* (2003).

[20] Hashem, T., Kulik, L., and Zhang, R. Countering overlapping rectangle privacy attack for moving knn queries. *Inf. Syst. 38*, 3.

[21] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. SSS. 2001.

[22] Huang, M.-S., and Narayanan, R. M. Trilateration-based localization algorithm using the lemoine point formulation. *IETE Journal of Research 60*, 1 (2014), 60–73.

[23] Jia, Y., Dong, X., Liang, Z., and Saxena, P. I know where you've been: Geo-inference attacks via the browser cache. In *W2SP 2014*.

[24] Kullback, S., and Leibler, R. A. On information and sufficiency. *Ann. Math. Statist. 22*, 1 (1951), 79–86.

[25] Li, M., Zhu, H., Gao, Z., Chen, S., Yu, L., Hu, S., and Ren, K. All your location are belong to us: Breaking mobile social networks for automated user location tracking. In *MobiHoc* (August 2014).

[26] Lin, Z., Foo Kune, D., and Hopper, N. Efficient private proximity testing with gsm location sketches. In *Financial Cryptography and Data Security*. 2012.

[27] Marathe, M., Breu, H., Hunt, H. B., Ravi, S. S., and Rosenkrantz, D. J. Simple heuristics for unit disk graphs. *NETWORKS 25* (1995).

[28] Marforio, C., Karapanos, N., Soriente, C., Kostiainen, K., and Capkun, S. Smartphones as practical and secure location verification tokens for payments. NDSS'14.

[29] Marquardt, D. W. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal of Applied Mathematics 11*, 2 (1963), 431–441.

[30] Mascetti, S., Bertolaja, L., and Bettini, C. A practical location privacy attack in proximity services. In *MDM* (2013).

[31] Mascetti, S., Bettini, C., Freni, D., Wang, X. S., and Jajodia, S. Privacy-aware proximity based services. In *MDM* (2009).

[32] Masuyama, S., Ibaraki, T., and Hasegawa, T. Computational complexity of the m-center problems on the plane. *IEICE TRANSACTIONS* (1981).

[33] Minami, K., and Borisov, N. Protecting location privacy against inference attacks. WPES '10.

[34] Narayanan, A., Thiagarajan, N., Hamburg, M., Lakhani, M., and Boneh, D. Location privacy via private proximity testing. In *NDSS '11*.

[35] Nieberg, T., and Hurink, J. A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs. In *WAOA* (2006).

[36] Polakis, I., Volanis, S., Athanasopoulos, E., and Markatos, E. P. The man who was there: validating check-ins in location-based services. In *ACSAC 2013*.

[37] Puttaswamy, K. P. N., and Zhao, B. Y. Preserving privacy in location-based mobile social applications. HotMobile '10.

[38] Qin, G., Patsakis, C., and Bouroche, M. Playing hide and seek with mobile dating applications. In *IFIP SEC '14*.

[39] Shokri, R., Theodorakopoulos, G., Le Boudec, J.-Y., and Hubaux, J.-P. Quantifying location privacy. In *IEEE Security and Privacy '11*.

[40] Shokri, R., Theodorakopoulos, G., Troncoso, C., Hubaux, J.-P., and Le Boudec, J.-Y. Protecting location privacy: Optimal strategy against localization attacks. CCS '12.

[41] Šikšnys, L., Thomsen, J. R., Šaltenis, S., Yiu, M. L., and Andersen, O. A location privacy aware friend locator. In *SST '09*.

[42] Theodorakopoulos, G., Shokri, R., Troncoso, C., Hubaux, J.-P., and Le Boudec, J.-Y. Prolonging the hide-and-seek game: Optimal trajectory privacy for location-based services. In *WPES '14*.

[43] Wang, Y.-C., Hu, C.-C., and Tseng, Y.-C. Efficient deployment algorithms for ensuring coverage and connectivity of wireless sensor networks. In *Wireless Internet, 2005*.

[44] Yang, Z., Zhao, Y., Liu, Y., and Xu, Y. Human mobility enhances global positioning accuracy for mobile phone localization. *IEEE TPDS 99* (2014), 1.

[45] Yu, B. Assouad, fano, and le cam. In *Festschrift for Lucien Le Cam*, D. Pollard, E. Torgersen, and G. Yang, Eds. Springer New York, 1997, pp. 423–435.

[46] Zhao, X., Li, L., and Xue, G. Checking in without worries: Location privacy in location based social networks. In *INFOCOM '13*.

[47] Zhong, G., Goldberg, I., and Hengartner, U. Louis, lester and pierre: Three protocols for location privacy. In *PETS '07*.

APPENDIX

A. APPENDIX

A.1 Expected Distance Lemma

LEMMA 8. *Let G be a square with edge length ℓ and consider the centroid p_c of G. Let $p_r \in \mathbb{R}^2 \cap G$ selected uniformly at random. Then,*

$$\mathbb{E}[\mathrm{dist}(p_r, p_c)] \approx 0.3825978 \cdot \ell$$

PROOF. Consider a square of side ℓ which, without loss of generality, is centered at $(0, 0)$. Let $p_r = (X, Y)$ where X, Y are random variables and also let $D(x, y) = \sqrt{x^2 + y^2}$. The distance of p_r from the centroid $p_c = (0, 0)$ of the square is given by

$$\mathrm{dist}(p_c, p_r) = D(X, Y) = \sqrt{X^2 + Y^2}$$

If $d(x, y)$ is the probability mass function for $D(x, y)$, since the random variables X, Y are independent, it holds

$$d(x, y) = d(x)d(y) = (1/\ell)(1/\ell) = \frac{1}{\ell^2}$$

where $d(x), d(y)$ are the probability mass functions for the distances on the x and y axes respectively. Therefore, the expected value of the distance is

$$\begin{aligned}
\mathbb{E}[\mathrm{dist}(p_c, p_r)] &= \int_{-\ell/2}^{\ell/2} \int_{-\ell/2}^{\ell/2} d(x, y) D(x, y)\, dx\, dy \\
&= \int_{-\ell/2}^{\ell/2} \int_{-\ell/2}^{\ell/2} \frac{1}{\ell^2} \sqrt{x^2 + y^2}\, dx\, dy \\
&\approx 0.3825978 \cdot \ell
\end{aligned}$$

$\square$