Pinpointing User's Exact Locations by exploiting People Nearby Feature in apps

Introduction:

People Nearby Feature is a feature that shows users how far other users also using the app are from their location. Previously this feature was exploited in the Telegram app, to pinpoint a victim's exact distance by spoofing one's latitude and longitude. We use this feature in *Skout*, *Meetme* and *Plenty of Fish* to approximate the location of those other users within a radius of 150m. For our project, we developed a python script that runs inside MITM proxy and outputs a set of 8 coordinates that are equally distant from the victim's location. The center of the polygon containing these 8 points would give us the approximate location of the victim. Man In The Middle(MITM) Proxy is a proxy that can be used to intercept, inspect, modify and replay web traffic. We use it to run python scripts that modify requests and responses, replay them later and run our algorithm.

Implementation:

Algorithm 1 (To get the minimum distance):

1. Extract the distance of the victim from the attacker's location (r1).
2. Draw a circle with radius = r1
3. Pick a random point on the circle
4. Find the distance of the victim from that point (r2)
5. Draw a circle with radius = r2
6. Find the intersection point of the two circles
7. Choose the point that is closer to the victim (r3)
8. Draw a circle with radius = r3
9. Repeat this process until we hit the minimum distance

Algorithm 2 (To narrow down the user's location):

1. Given the circle with the minimum distance
2. Shift the circle's center in the following directions : 0,45,90,135,180,225,270,315,360
3. In each direction increment the distance by 50m, keep incrementing the distance until the distance extracted from the app is greater than the minimum distance limit set by the app
4. Keep track of the previous point, and save the previous point once the distance extracted is greater than the minimum distance limit set by the app
5. Use the saved points to draw circles with minimum distance
6. Draw a polygon by joining all the points
7. Center of that polygon gives the approximate location of the victim.

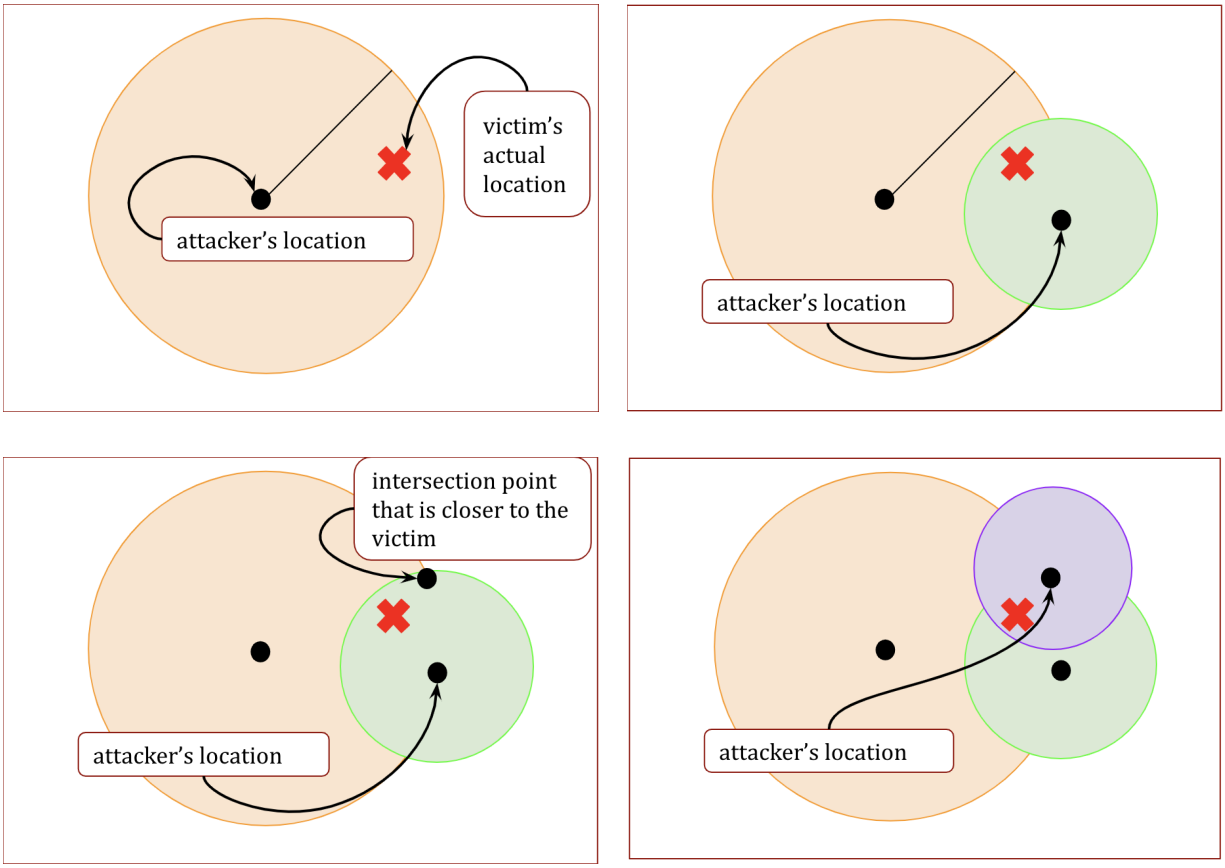

Figure 1: Pictorial representation of Algorithm 1

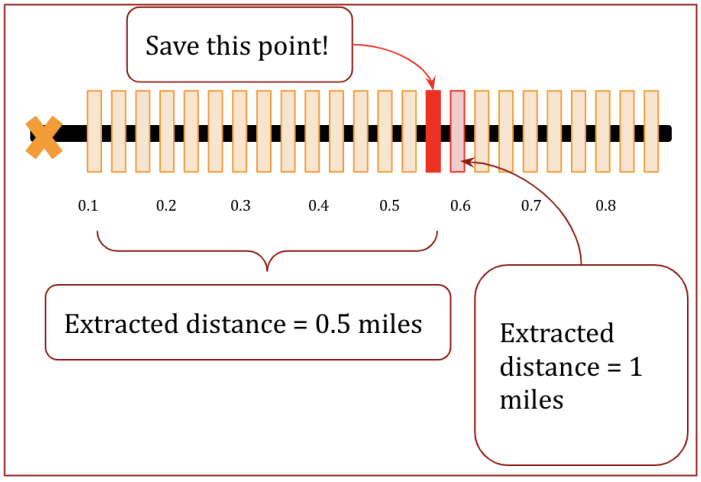

Figure 2: Narrowing down the user's location in (0 degrees direction).
Yellow cross mark represents the center of the circle with the minimum distance.
Repeat the same process for 7 other directions.

Challenges:

Challenge 1: Passing Information from Response to the Algorithm:
MITM allows you to run a piece of code upon receiving a response from the server. That particular code will run for each response that MITM receives. So we cannot implement our Algorithm in this code because we need to spoof our location multiple times during the procedure. So the challenge here was to figure out how we can run our algorithm outside the response capture code while passing the information from the response to the algorithm.

Challenge 2: Synchronizing the events in the Algorithm:
After sending the updated location request to the server, we need to wait for the server to update the attacker's location before we send the distance information request. Similarly, we need to wait after sending the distance information request till we get the updated distance of the victim in the response.

Our Solution for Challenge 1 and 2:
Our Algorithm runs asynchronously. It changes the location parameters in the request headers and replays the request to the server. Server's response is captured by MITM in a different thread. We created two shared variables between our algorithm and the response thread. One variable keeps track of the victim's distance and the other is a semaphore. Our algorithm awaits on the shared semaphore after sending the location update request to the server. The response threads signals the shared semaphore once it gets the response from the server that the location has been updated. Algorithm proceeds with sending the second request for the location information of the victim and awaits on the shared semaphore again. Response thread gets the location information of the victim and updates the distance in the shared distance variable and signals the shared semaphore. Algorithm gets the signal and proceeds with its work. This procedure repeats for every time the location is being updated.

Challenge 3: Server delay in updating the response.
During our testing we noticed that there was a lag between the *update location* request and *actual distance update* at the server end. We were getting the *update location* response back from the server that said that the location was successfully updated but when we asked for the distance information of the victim, the server sent the distance information based on the previous location.

Our Solution to challenge 3:
We added sleeps after each *location update* request to give the server some time to update the location on its end and we asked the server multiple times (4 times) for the distance information before we updated the distance in our algorithm.

Challenge 4: Minimum distance displayed by the app
While analyzing the app's data and how the distance information is handled by the app we noticed that all the chosen apps had a limit on the minimum distance it would display if you are too close to

the victim. For example, Skout shows 0.5 miles if you are less than or equal to 0.5 miles away from the victim. This limits the accuracy with which you can compute the location of the victim.

Actual Distance	Displayed Distance
0 - 0.5 mile	0.5 mile
0.6 - 0.9 mile	1 mile
1.0 - 1.9 mile	1.(something)

Table 1: the actual distance versus the distance displayed by Skout

Our Solution to Challenge 4:
When we get less than 0.5 miles away from the victim we stop running our **first Algorithm** and start running our **second algorithm**. Given the circle with the minimum distance, we shifted this circle's center in 9 directions. Throughout the computation we keep track of the current and the previous latitude and longitude coordinates. We start going in one direction and increment our distance by 50 meters until the app shows that the distance from the victim is more than the minimum distance limit set by the app. At this point we are sure that the previous point is exactly 0.5 miles from the victim's location. We do this in all the directions. The algorithm is described in the Implementation Section in detail.

Evaluation:

We conducted multiple experiments by spoofing both the victim (Muzzamil) and attacker's (Jyotsna) location. Using our approach we accurately identified the victims location within 50 meters radius, in the best case there was 16x improvement compared to the minimum distance method. In the worst case, we accurately identified the victims location with 150 meters accuracy, this is still a 4.5x time improvement over the minimum distance method. The accuracy depends on the jumps taken in each direction (distance by which we increment) as well as the number of directions picked. For the results shown in this section, we incremented the distance by 50m in 9 different directions. We also conducted experiments on unknown users on Skout. We were able to compute a circle of radius 100m, but there is no way to verify how accurate their location information is.

Case Study 1:
Experimental setup:

- Victim's Location: Pujan's house
- Attacker's Location: Jyotsna's house

We were able to successfully identify Pujan's location within a circle of radius equal to 100m.
The blue circle represents Pujan's location, which falls inside our computed circle of radius 100m.

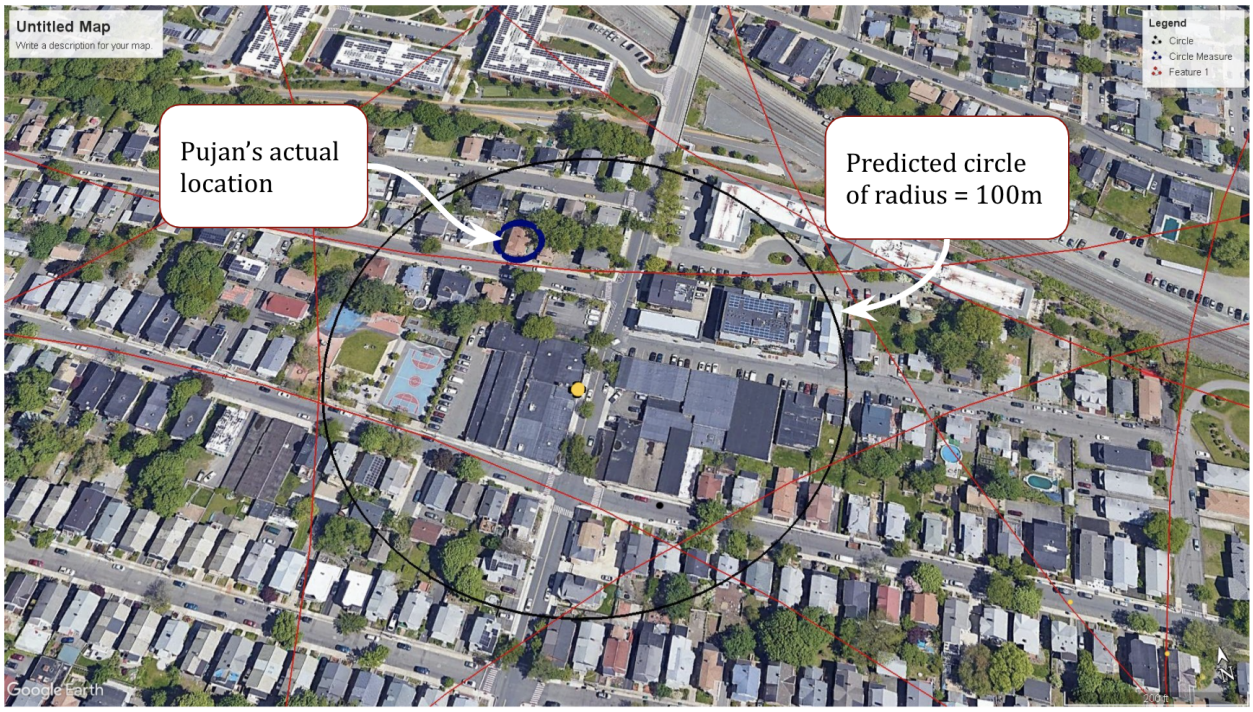

Figure 3: Pujan's Location is within our computed circle of radius = 100m

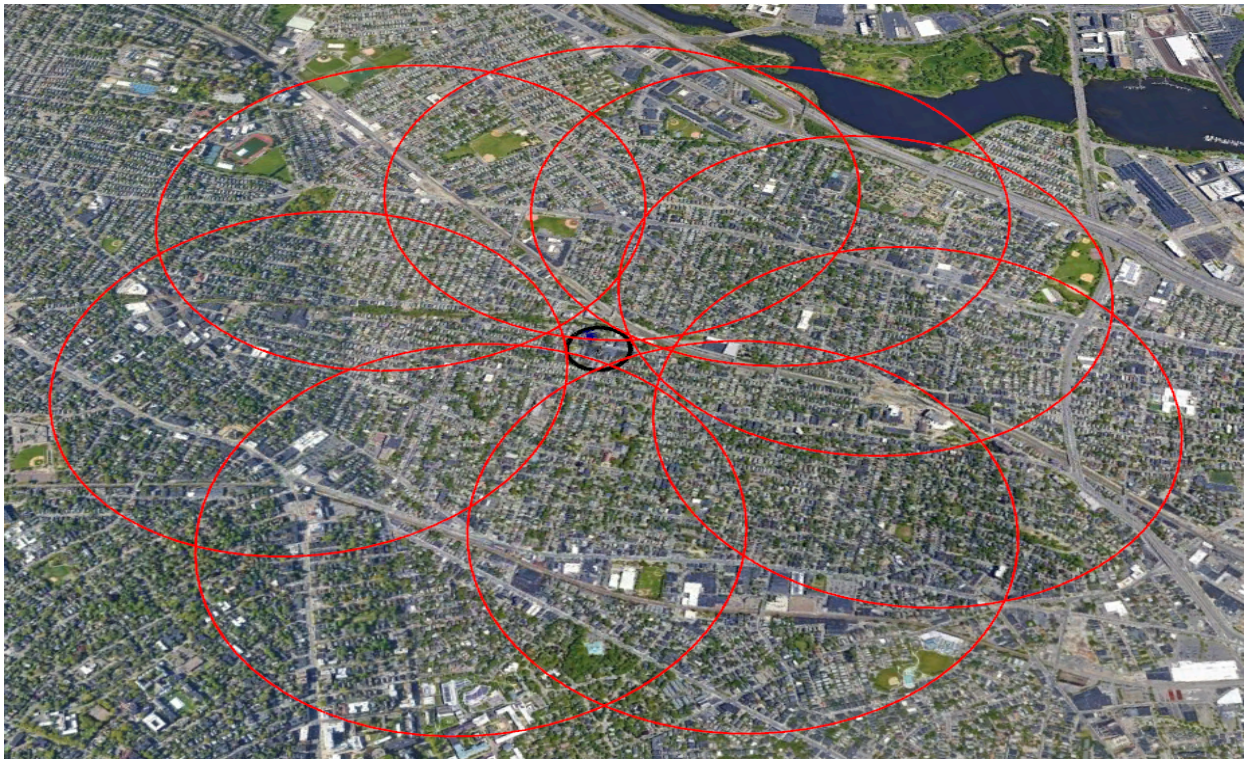

Figure 4: The 9 circles drawn around the victim's location, the centers of these 9 circles are equidistant from the victim's location.

Case Study 2:
Experimental setup:

- Victim's Location: Muzammil's house
- Attacker's Location: Jyotsna's house

We were able to successfully compute Muzammil's location within a circle of radius equal to 100m.

Discussion & Limitations:

1. *Manual Work:* Identifying relevant requests for location and distances required analysing each individual request (and there were 100s of requests) that were sent from the server. Also there were multiple requests that sent the location information, but making changes to these requests did not update the location information on the server side.
2. *Obfuscation in the app data:* We had to analyse the network data of ~10 apps, before we selected 3 apps that could be automated for our project.
3. *Accuracy:* Our approach is limited by the accuracy of the distance (in miles usually) calculated by the server. Most servers have an uncertainty of 0.1 mile in the distance.
4. *Improvement In Accuracy:* We can improve our accuracy by taking smaller intervals (in terms of the distance) and considering more than 9 directions between (0 - 360), but this would take more time because of the server lag. It took more than 30 mins when we incremented the distance in 10m intervals and considered 9 directions . Our current approach with 50m intervals and 9 directions takes 15 mins.

GitHub Link:

https://github.com/muzammilLUMS/Vulnerability-Analysis