The Perils of User Tracking Using Zero-Permission Mobile Apps

Sashank Narain, Triet D. Vo-Huu, Kenneth Block, and Guevara Noubir | Northeastern University

Users' traveled routes and locations can be exploited by permissionless access to smartphone motion sensors. Extensive simulations for 11 cities and real experiments for two cities show that a significant number of users are vulnerable to tracking by seemingly innocuous apps.

The mobile revolution has profoundly changed how we share information and access services. Modern smartphones are equipped with sophisticated sensors that have redefined how users interact with the environment. Despite their benefits, they've opened the door to numerous privacy invasion attacks. One such attack is the leakage of location information to track users, discover their identity, and identify home and work locations.

A simple method of leaking users' location is accessing their smartphone's location services that rely on GPS, Wi-Fi, or cellular signals. To mitigate such leakage, mobile OSs (for example, Android and iOS) allow users to manage permissions and access sensitive resources and information. For example, Android mobile apps must request permission to access location information, giving users the opportunity to decline. This is a good start, but many users are still careless about checking such permissions. This is illustrated by the recent charges by the US Federal Trade Commission against the "Brightest Flashlight" app for deceiving consumers by sharing location information without their knowledge.[1] With a 4.7-star rating and more than one million users, this app is just one example of many seemingly innocuous applications with questionable privacy practices.

Careful users can easily determine that a flashlight app shouldn't access their location information. Also, users unwilling to disclose their location can simply disable location services on their device. However, it's much harder to detect indirect location leakage from side channels, such as the gyroscope, accelerometer, and magnetometer embedded in most modern smartphones. Currently, any Android app can access these sensors without permissions or any visual cue to the user. There is no mechanism to disable these sensors because many system services, for example, orientation, use them to perform their functions. As such, exploiting these sensors has become an attractive target for privacy attacks.[2–6]

We investigated the threat of tracking users' routes and locations using zero-permission smartphone sensors, without location services. Our goal is to raise awareness about the dangers of these attacks in mobile devices and implement mitigation techniques to prevent them.

We focused on a scenario in which users are traveling in a vehicle on public roads. With simulations, we show that for most cities, it's possible to generate a short list of 10 routes containing the user's traveled route with probability higher than 50 percent. The inference of traveled route can easily lead to inferring the user's

 Copublished by the IEEE Computer and Reliability Societies

home and workplace locations, and further information about the user's identity can be derived from the town's public database.

Attack Scenario

We focus on a very simple and common attack scenario. An adversary, with the intention of tracking users, uploads an attractive and seemingly harmless-looking app on a global repository, such as the Google Play Store. Once uploaded, this app becomes available for download to millions of users worldwide. When users download and use this app, it performs its intended function but also starts a malicious service in the background to detect when a user sits in a car. This malicious service starts recording the sensor data from the accelerometer, gyroscope, and magnetometer while the user is driving. From this sensor data, the app derives driving information such as turn angles, route curvature, acceleration, and heading and then uploads this information to a colluding server. As the upload size is small (roughly 80 Kbytes per hour, or slightly above 20 bytes per second), degradation of network performance due to malicious traffic is negligible. Using publicly available geographic area attributes, the adversary can learn the actual routes taken without the need of location services.

Approach

We developed a location-tracking framework to address many challenges that undermine the feasibility of this attack and to optimize tracking by creating architecture blocks that address every challenge, modeling location tracking from sensors as a graph theoretic problem, and developing efficient search algorithms that maximize the probability of finding the traveled route in a small set of results. Here, we discuss the challenges, architecture, and the design of our graphs and algorithms.

Challenges

The challenges that undermine the feasibility of the attack include the following:

- *Area size:* The geographic area's size impacts the estimation of the route. There are billions of possibilities for a route even in small cities like Waltham, Massachusetts.
- *Road similarity:* Similarity in roads also impacts the estimation of the route, especially in grid-like road structures like Manhattan's.
- *Noisy sensor data:* Sensor data quality is key for high accuracy. However, today's smartphones are equipped with low-cost sensors that don't guarantee high accuracy. These sensors are also strongly impacted by the environment—for example, accelerometers spike due to speed bumps or potholes, and magnetometers are influenced by magnets in fans, speakers, and other electromagnetic devices.
- *Driving patterns:* Drivers frequently vary their speed or switch lanes to overtake others. These actions induce noise in the sensors as spatial distortions.

Architecture

The architecture comprises two main actors: a smartphone that collects sensor data and a server that processes this data, searches it, and generates a ranked list of potential routes. The server framework is modeled using graph theory: the road network of each different area is converted to a searchable graph structure, and the user's route is searched on the graph by our algorithms that are designed to maximize the likelihood between the actual route and the route inferred from sensor data. The architecture blocks are described below.

- *Preparation:* Road information from public map resources are extracted and converted to graph structures. This is a one-time initialization step; the graph can be reused for all subsequent attacks.
- *Sensor data collection:* Sensor data is recorded by the app and uploaded to a colluding server. This step detects user movement from accelerometer data and triggers sensor recording exclusively during vehicle movement.
- *Data processing:* The uploaded sensor data is processed and analyzed to derive the user trace consisting of turn angles, curvatures, heading, and time stamps.
- *Search:* The search algorithm is run on the user trace, and a ranked list of matching routes is produced.

Graph Construction

Our graph is constructed from real map information obtained from OpenStreetMap (OSM; www.openstreetmap.org), which provides free access to accurate and detailed global road network information. Speed limit information was extracted from the Nokia HERE platform (maps.here.com), which we found to be more accurate. We explain our graph construction using the small hypothetical road network shown in Figure 1a.

The hypothetical road network is decomposed into a set of atomic sections. An atomic section is a one-way road section between two intersections that doesn't contain sharp curves or turns (that is, angle change of more than 30°). $\overrightarrow{BA}$, $\overrightarrow{BC}$, $\overrightarrow{BD}$, $\overrightarrow{CB}$, $\overrightarrow{DB}$, $\overrightarrow{JB}$, $\overrightarrow{EJ}$, $\overrightarrow{FJ}$, $\overrightarrow{FG}$, $\overrightarrow{TF}$, and $\overrightarrow{HT}$ are examples of atomic sections. The curvature of the section, the fastest time from start to end, and the heading direction become the attributes of each section. $\overrightarrow{FG}$ is an example of a curved atomic section, $\overrightarrow{BA}$ has a heading of 270° (west), and $\overrightarrow{BC}$ has a heading of 0° (north). This curvature and heading information is extracted from the map coordinates, and

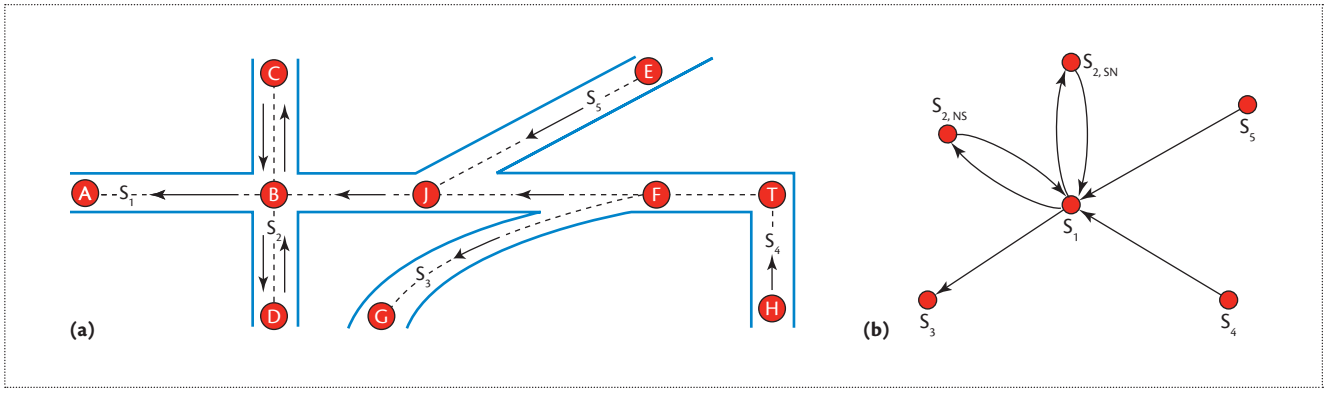

Figure 1. Example of a road network and its mapping to graph. (a) Connections are created when a road bisects (B), furcates (F), joins another road (J), or turns in a different direction (T). The created atomic parts are $\overrightarrow{BA}$, $\overrightarrow{BC}$, $\overrightarrow{BD}$, $\overrightarrow{CB}$, $\overrightarrow{DB}$, $\overrightarrow{JB}$, $\overrightarrow{EJ}$, $\overrightarrow{FJ}$, $\overrightarrow{FG}$, $\overrightarrow{TF}$, and $\overrightarrow{HT}$. (b) Graph construction. Every one-way road segment s_1, s_3, s_4, and s_5 is represented by one vertex, while two vertices $s_{2,NS}$ and $s_{2,SN}$ are created for the north–south (NS) and south–north (SN) directions of road segment s_2, respectively.

travel time is calculated from the section's maximum speed limit.

Atomic sections are chained to form maximal-length segments if they form a straight or slightly curved road. Each section's attributes (curvature, travel time, and heading) are merged and added as an attribute of the segment. These segments can contain many intersections connecting to other segments. For example, $s_1(\overrightarrow{TFJBA})$, $s_{2,NS}(\overrightarrow{CBD})$, and $s_{2,SN}(\overrightarrow{DBC})$ are three segments intersecting at B, where NS (north–south) and SN (south–north) indicate heading directions.

Connections are formed between segments that share an intersection, and turning from one segment to another is permitted. The turn angle becomes an attribute of the connection. Intersection B is an example of connection between segments s_1 and $s_{2,NS}$ (90° left turn) and between s_1 and $s_{2,SN}$ (90° right turn).

The segments form the graph vertices V, and the connections form the edges E of the directed graph G. Figure 1b shows the graph we constructed for our hypothetical road network. Graph and sensor data are provided to the search algorithms for calculating the user's actual route.

Sensor Data Processing

Sensor data provides useful information about a user's route. Gyroscopes are most reliable because they reveal turn and curvature information, which are traceable on a public map. Accelerometers and magnetometers can be unreliable because of environmental factors like traffic, road conditions, or proximate magnetic fields. These environmental perturbations are difficult to predict and compensate for. Intuitively, we use the gyroscope as our primary scoring sensor and the accelerometer and magnetometer for supporting data, such as idle time and heading estimate, to refine the results.

The gyroscope provides a sequence of 3D vectors where each axis reports the rate of angular change the smartphone experienced. The turns and curvatures are calculated by integrating these rates over time. Figures 2a and 2b illustrate an experimental route and its corresponding angle sequence. The large changes in angle trace indicate turns (negative for right and positive for left), and minor variations indicate road curvature.

Error compensation. Gyroscopes suffer from drift where their readings drift away instead of reporting 0 on each axis (when idle). Figure 2b shows a large drift in the *y*-axis. This drift causes additional errors in the calculated turn angles and curvature. Their complete removal requires computation-expensive sensor fusion algorithms. Estimating a unit amount of drift on each axis when a vehicle is idle and subtracting this from the entire trace reduces the drift. Figure 2c shows the reduced drift in the *y*-axis.

Rotation. Users can place their smartphone in any orientation in the car. We virtually rotate the gyroscope data such that its *z*-axis points upward perpendicular to the Earth's surface. This is done by using gravity measurement from each accelerometer axis and computing the appropriate rotation matrices. Thus, the turn and curvature information is entirely represented by the *z*-axis (Figure 2d), making it easier to extract the information.

Trace extraction. The gyroscope trace between two turns forms the curvature of vertices V, and the turns form the edges E of the graph. The vertices carry additional information such as idle time from the accelerometer and heading from the magnetometer. We collected this info by doing the following:

- *Extracting turn angles and curvature:* The turn angles and curvature are extracted from the gyroscope's z-axis using predefined thresholds. We label a slope as a turn when it's more than 30° within a short time; otherwise, it's considered as curvature.
- *Detecting idle time:* Idle time is calculated as the sum of time frames where the accelerometer reports near-zero magnitudes in all three dimensions. These values are considerably larger during motion. The detection of idle states helps us better estimate the travel time and improve the attack performance.
- *Estimating heading:* The heading is extracted only when the magnetometer doesn't experience strong magnetic fields—that is, reported field strength is comparable to the region's field strength ($30–50\mu T$ for the Northeast US).

The Search Algorithm

The search algorithm traverses the entire graph for every route and maintains a list of scored potential routes that have a high probability of matching the recorded driving trace. The algorithm outputs two ranked lists—individual ranks and cluster ranks. The individual rank gives the rank of the exact route in the search results, whereas cluster rank gives the rank of a group of similar routes that match the exact route. Clustering might group routes with the same end points, ignoring different roads in between, or routes that converge to the same start/end area, for example, roads going from/to a residential complex or office. This could give an adversary more confidence in certain areas than the individual rank.

Objective. The algorithm's objective, given a route with N turns, is to find a sequence of turns $\theta = (\theta_1, \ldots, \theta_N)$ in graph G that maximizes the probability of matching θ given the observation of gyroscope turns $a = (a_1, \ldots, a_N)$. This probability, denoted as $P(\theta \mid a)$, can be written as

$$P(\theta|\alpha) = \frac{P(\theta,\alpha)}{P(\alpha)} = \frac{P(\alpha|\theta)P(\theta)}{P(\alpha)}.$$

The a priori probability $P(\theta)$ is equiprobable—that is, all routes have the same probability of being taken by a user. This presents a worst-case scenario and gives a lower bound on the performance. In addition, $P(a)$ is independent of θ because it's the probability of measurement a without conditioning on θ. Therefore, maximizing $P(\theta \mid a)$ becomes equivalent to maximizing the probability of $P(a \mid \theta)$ alone.

The gyroscope turns contain a random amount of noise n, yielding the angle $a = \theta + n$. Through experimentation, which we discuss later, we see that this noise can be approximated by an N-dimensional zero-mean normal distribution $N(0, \sigma)$ with standard deviation σ. $P(a \mid \theta)$ can be rewritten as

$$P(\alpha|\theta) = P(n = \alpha - \theta) = (2\pi\sigma^2)^{-\frac{N}{2}} \exp\left(-\frac{\|\alpha-\theta\|^2}{2\sigma^2}\right),$$

where $||.||$ indicates a vector's L_2 norm. The $(2\pi\sigma^2)^{-N/2}$ is constant for a fixed N and σ; therefore, maximizing

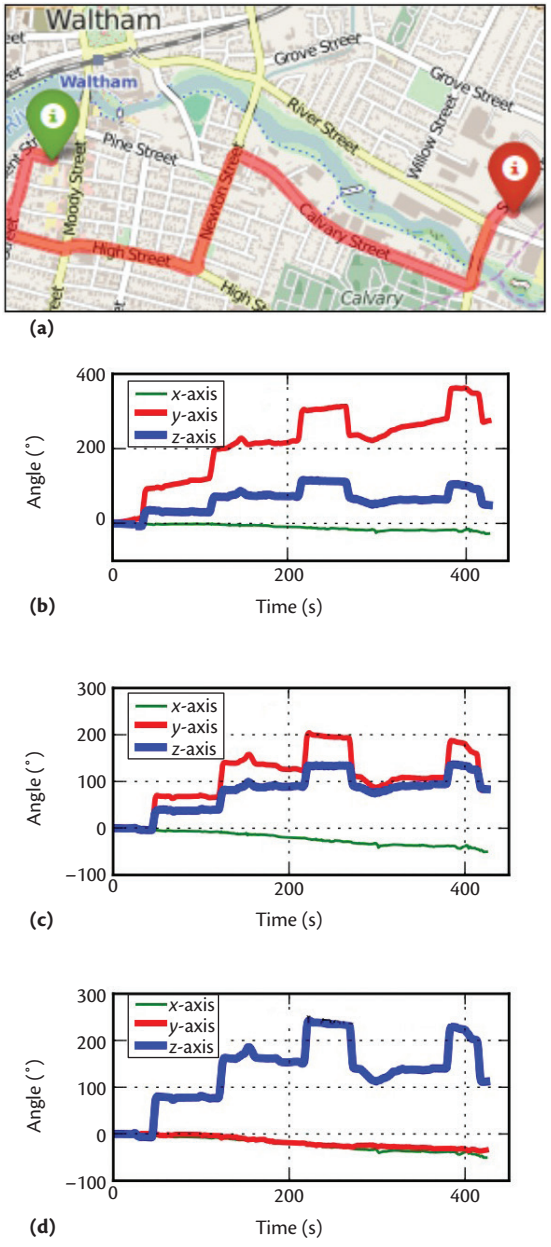

Figure 2. Error compensation steps for gyroscope data: (a) experimental route, (b) recorded gyroscope data, (c) calibrated gyroscope data, and (d) rotated gyroscope data.

$P(a \mid \theta)$ is equivalent to minimizing $||a - \theta||$. The optimal route-tracking solution becomes

$$\theta^* = \arg\min_{\theta \in G}\left(\left\|\alpha - \theta\right\|\right).$$

Algorithm. A high-level pseudocode for the algorithm follows. First, consider each vertex (that is, road segment) as a potential starting point. Iterate through every edge (that is, intersection), and then iterate through each potential route. In doing so, extract all forward connections, eliminate all unlikely connections (filter), score other connections, and add to the route's score. Finally, pick the top-scoring paths.

The algorithm iterates through every potential route for each intersection and extracts all the forward connections. Unlikely connections are filtered out, and the remaining are scored and added to the previous path score. The algorithm picks only a certain number of top-scoring paths after each iteration. The filtering and selection of top paths are essentially a tradeoff between searching speed and accuracy.

Filtering. Connections that don't satisfy certain flexible thresholds are eliminated. These represent turns or segments that are highly unlikely to be traversed by a user because the turn difference between connection and reported is too large (≥60°) or the direction of a road extracted from the map deviates too much from the reported data (for example, the map indicates east–west direction, whereas the reported data suggests south), or it's impractical to reach a connection in the reported time (for example, a connection takes minimum five minutes to reach while reported time is 20 seconds).

Scoring. The score for every connection is calculated based on turn angle, curvature, and travel time. For each new intersection *i*, scores of individual features are added to the route score. The final route score S_{Final} is the sum of scores for all N intersections:

$$S_{Final} = \sum_{i=1}^{n}\left(s_\alpha + s_c + s_t\right)_i.$$

The score for turn angle s_a and travel time s_t is calculated as the absolute value of difference between the reported (a and t) and connection (θ_i and t_i) values, multiplied by the turn angle weight ω_a and time weight ω_t, respectively:

$$s_\alpha = \omega_\alpha * |\alpha - \theta_i|;$$
$$s_t = \omega_t * |t - t_i|.$$

The curvature scoring is a bit more complex. Here, we divide the connected segment curvature into T subsegments and denote the ith intersection curvature as a sequence of subturn angles ($\vartheta_1, \ldots, \vartheta_T$). Similarly, the curvature trace is derived from the gyroscope data as ($a_1, \ldots, a_T$). The curvature score s_c is calculated as the normalized distance between two curvature sequences, multiplied by the curve weight ω_c:

$$s_c = \omega_c * \frac{1}{T}\sum_{k=1}^{T}\left|\alpha_k - \vartheta_k\right|.$$

The weights are assigned based on the data source's accuracy. We assigned higher weights to turn angle and curvature because the gyroscope is more reliable than other sensors. They can further be adjusted based on whether a city has more curved roads or unique turns. The travel time is assigned a lower weight, as traffic in any city can be unpredictable.

The L_2 norm is theoretically optimal for Gaussian distributions, but the L_1 norm worked better for us in real experiments because computing the L_1 norm reduces system overhead and improves search time. In addition, the gyroscope errors weren't truly Gaussian, and squaring large sparse errors in L_2 amplified those errors.

Evaluation

We evaluated the attack potential using simulations from several cities and real experiments from Boston and Waltham. Here, we report the results for these experiments, but first we justify the attack's feasibility based on gyroscope accuracy and algorithm performance. We also provide some intuition about the selection criteria for the simulation cities.

Gyroscope Accuracy

Gyroscope accuracy has the largest impact on the attack's feasibility. We measured the accuracy for four smartphones by calculating the error between the turn angles obtained from real experiments and the corresponding turn angles from the maps. The error distribution for each phone showed that they closely followed a normal distribution. All phones—that is, HTC One M7, LG Nexus 5, LG Nexus 5X, and Samsung Galaxy S6—reported a near-zero mean and almost equal standard deviations σ of 7.07°, 7.89°, 6.40°, and 7.51°, respectively. More than 95 percent of errors were below 10°, which is fairly accurate.

Algorithm Performance

Algorithm performance is another factor that determines attack feasibility. A longer search time would substantially degrade attack performance due to resource limitations and restrict the attack to specific areas.

Table 1. Cities used for evaluation with their characteristics.

City	No. of segments/vertex	No. of turns/edge	Mean μ turn (°)	Standard deviation σ turn (°)
Atlanta	10,529	25,557	88.73	17.58
Berlin	4,708	19,752	88.21	19.87
Boston	8,080	22,149	89.69	20.52
Concord, Massachusetts	3,049	6,467	88.13	29.58
London	9,468	21,968	87.83	20.38
Madrid	10,012	30,144	86.41	25.13
Manhattan	1,033	3,699	89.23	17.81
Paris	6,744	11,204	86.35	26.26
Rome	9,408	20,577	85.98	26.15
Sunnyvale, California	5,592	12,302	88.59	16.00
Waltham, Massachusetts	3,366	9,437	88.93	20.53

We didn't perform a formal benchmark of our algorithm, but the simulations provided enough information to determine performance. For example, searches on a virtual machine (VM) with 16 threads at 2.93 GHz and 32 Gbytes RAM took 2.2 s on average for a large city like Atlanta, and only 0.4 s on average for a small city like Concord, Massachusetts. Our simulations took ≈21 hours (≈0.85 s per route) for 88,000 routes, which means that adversaries would need only 10 similar VMs running in parallel to search 1 million routes (users) in a single day. This gives them the capability to track millions of users without detection.

City Selection

We chose 11 cities, based on their diverse sizes and road structure, to study attack potential. Table 1 shows these cities along with characteristics such as size (segments/vertex $|V|$ and turns/edge $|E|$) and turn variance (mean μ turn and standard deviation σ turn).

A large city like Atlanta with high $|V|$ and $|E|$ values and low σ turn suggests a very low inference potential as its size significantly increases the search space due to large number of connections. Cities like Concord, Madrid, Paris, and Rome with a high σ turn suggest high inference potential as they have many unique turns that reduce the search space due to elimination of unlikely connections. Conversely, grid-like cities like Berlin, Manhattan, and Sunnyvale, with a low σ turn, suggest low inference potential. The other cities like Boston, London, and Waltham have a mix of gridlike and unique turns that suggests good inference potential.

Simulation Evaluation

We performed simulations for all 11 cities using scenarios that emulated real environments. We first generated a set of 2,000 simulation routes for each city, and added four error scenarios to each route. Then we ran a search for all 8,000 simulation routes.

Generation of simulation routes. We created simulation routes by first randomly choosing a number of turns $N \leftarrow \{3, ..., 10\}$ and then selecting a random set of $N + 1$ connected segments from the graph, such that all turn angles were between 30° and 150° and the time between turns was more than 10 s. No constraints were placed on the route length; they were between ≈0.5 km and ≈8.15 km, with an average length of ≈7.15 km.

Adding errors to simulation routes. The simulation routes represent an ideal, error-free scenario, so we defined additional error levels to simulate more realistic scenarios. In all error scenarios, the magnetic heading error was added by a uniform distribution between −90° and 90°, and the time error was added by a uniform distribution between a variable lower bound (β) and a fixed upper bound of 1.5. These bounds are a function of speed—for example, for a road with a maximum speed of 100 km/h, a lower bound of 0.1 implies driving at 10 km/h, and the upper bound of 1.5 implies driving at 150 km/h. We added the turn and curvature errors using a normal distribution with specific scenario-dependent standard deviation σ. We categorized performance scenarios as follows:

- *Ideal:* Noise-free scenario (upper-bound performance).
- *Worst:* $\beta = 0.1$, $\sigma = 10$. This scenario simulates heavy traffic and phones with less accurate gyroscopes.
- *Typical:* $\beta = 0.5$, $\sigma = 8$. This scenario simulates moderate traffic and current phones. The $\sigma = 8$ is slightly higher than the experimental value ($\sigma = 7.54$), implying a slightly harder attack.
- *Future:* $\beta = 0.5$, $\sigma = 6$. This scenario simulates

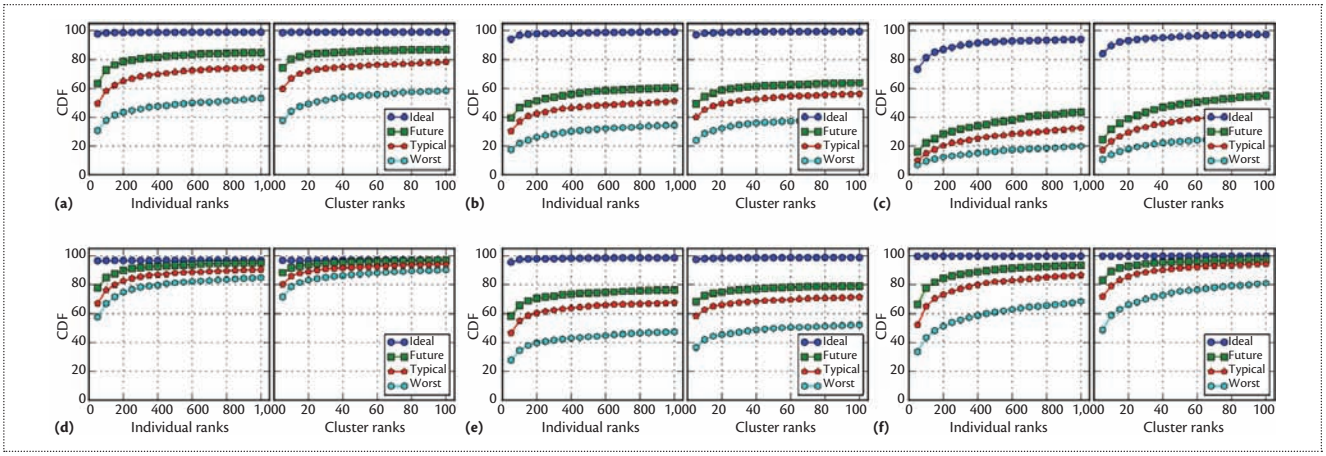

Figure 3. Attack performance showing cumulative distribution function (CDF) on simulated routes for six cities. (a) Sunnyvale: σ = 16.00, (b) Atlanta: σ = 17.58, (c) Manhattan: σ = 17.81, (d) London: σ = 20.38, (e) Boston: σ = 20.52, and (f) Waltham: σ = 20.53. (Graphs are arranged in ascending order of turn distribution σ.)

moderate traffic and future phones with more accurate gyroscopes.

Analysis of simulation results. We evaluated the simulation results using both individual rank and cluster rank metrics. The distance threshold we chose for clustering was Δ = 500 m as it typically covers a few blocks or apartment buildings. We chose time weight 0.1 to minimize the impact of unpredictable traffic but still assign a better score to the most likely connection among similar connections. The turn and curvature weights were each 2.5 to assign them a much higher priority than travel time. This weight was chosen among several potential weights as it yielded the optimal results for simulations in several cities. The same weights were used for every city to evaluate the attack using the same configuration for different city profiles.

Figure 3 shows the cumulative distribution function (CDF) of the individual and cluster ranks for six cities. The CDFs of the other cities and a more detailed analysis of the results can be found in an extended version of the article.[7] The results show that, in most cities, the exact route was in the top 10 for more than 50 percent and 35 percent of routes in the typical and worst scenario, respectively. The only exceptions were Atlanta, Berlin, and Manhattan. We also see that cluster ranks are only slightly higher than individual ranks because each cluster comprises a relatively small set of routes ($\approx$1 to 20 routes per cluster).

The cities with spread-out turns ($\sigma \geq 20$) showed higher attack potential because the high turn variance (σ) had a large positive impact on scoring. The results in Concord were the most successful because its many curvy roads and small size diversified the score and reduced the search space. There were some disparities between cities with similar σ, which can be attributed to road curvature. For example, Boston, London, and Waltham have a similar σ, but London shows very high potential as compared to the others. The reason is that Boston and Waltham have several grid-like residential areas, such as South Boston and Back Bay, that create confusion for routes passing through them. Waltham had better rankings than Boston primarily because it's much smaller in size. Similarly, results in Rome were much better than Madrid and Paris owing to the presence of many more curved roads.

The cities with grid-like roads ($\sigma < 20$) showed the lowest attack potential primarily because their fewer unique turns reduced the turn angle impact on the scoring. The results in Manhattan were lowest because most roads are straight and head north–south or east–west, reducing the curvature's and heading filter's impact. The results in Atlanta were low because the city's large size increased the search space to billions of possible routes. The results in Berlin were largely affected by absence of curved roads, reducing the curvature impact. Sunnyvale showed higher potential owing to the presence of more curved roads, despite the lowest turn variance.

Real Driving Experimental Results

To evaluate the real driving experiments, we collected sensor data from driving routes, processed the data for every route, and ran a search on all the processed data.

Collecting sensor data from driving routes. The real experiments comprised more than 70 unique routes each for Boston and Waltham carried out by four drivers. These routes emulated realistic scenarios, for example, travel between residential areas, shopping malls, office, or city centers. The shortest route taken was

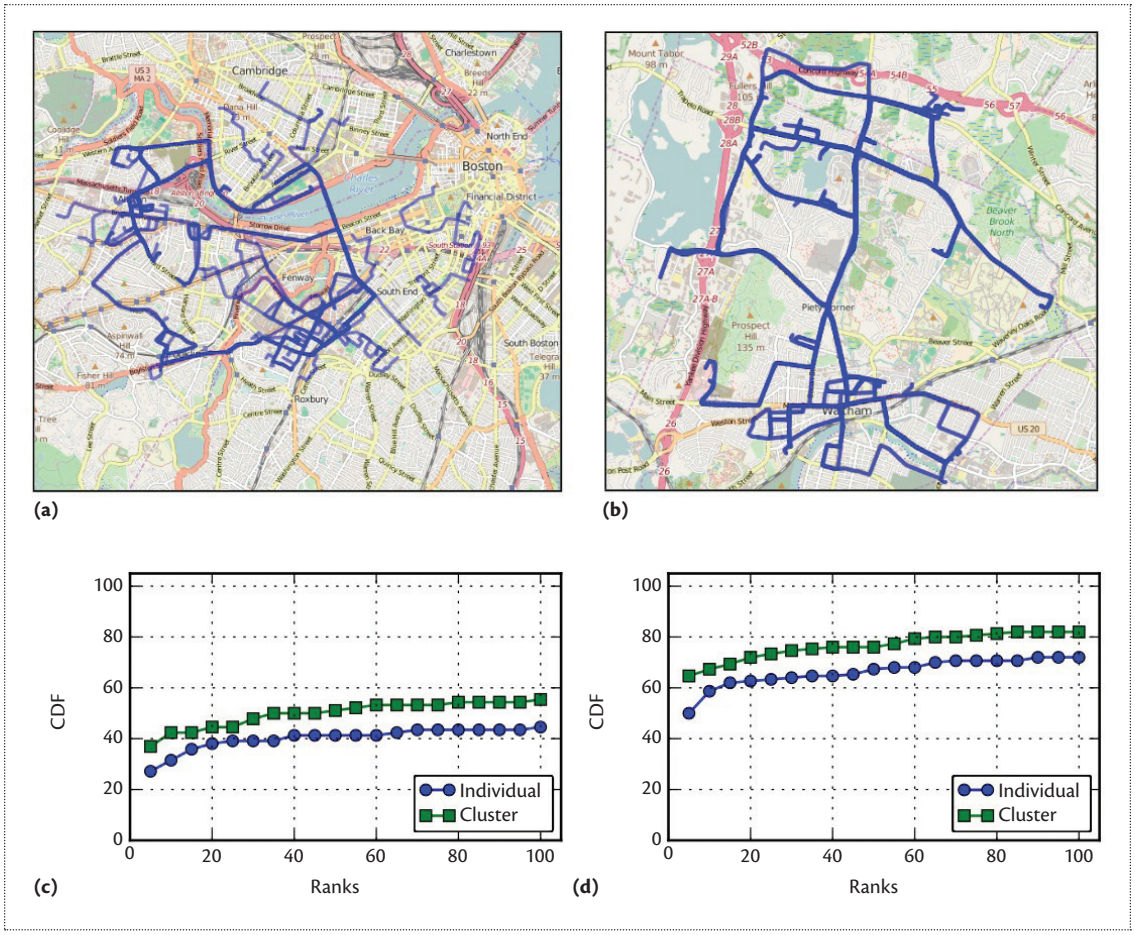

Figure 4. Attack performance for all traveled routes and their real experiment statistics: (a) GPS traces of all traveled routes in Boston and (b) Waltham, and attack performance in (c) Boston and (d) Waltham.

≈0.75 km and the longest ≈7.25 km, covering ≈980 km of driving in both moderate and heavy traffic. Four more routes were taken to consider scenarios of driving in a circle, taking many turns (≥20), and traveling longer distances (≥20 km). We also used these routes to test the system's stability. Figures 4a and 4b show the GPS traces for both cities (used only for ground truth comparison).

The drivers were requested to

- place the phone anywhere in the car in a fixed position,
- idle for about 10 s before driving, and
- take a minimum of three turns on their route.

These requirements allowed us to model typical scenarios in which a person places the phone in a stable position (for example, in a cup holder or mount) and then takes some time putting on seatbelts and adjusting the seat and mirrors. For this initial study, we didn't consider situations in which the vehicle starts by driving in reverse.

Analysis of experimental results. The evaluation metrics and thresholds were the same as the simulation. This time, we fine-tuned the scoring weights (turn, time, and curve) to 2.5, 0.1, and 3 for Boston and 2.25, 0.1, and 2.5 for Waltham. The turn weight for Waltham was reduced to 2.25 mainly to increase the impact of travel time, as Waltham is typically less congested and traffic is more predictable. The curve weight was higher than turn weight for both cities because they, especially Boston, have more curved roads than unique turns, implying that curvature might have a larger impact on the scoring than turns.

The CDF of the individual and cluster ranks are shown in Figures 4c and 4d. We see that roughly 50 percent of routes in Waltham and roughly 30 percent of routes in Boston are in the top five individual ranks. When the top one is considered (that is, exact route), the success probability reduces to 38 percent for Waltham and 13 percent for Boston, respectively. The gap between individual and cluster ranks is approximately 10 percent, which is almost like simulations. The number of routes per cluster is around two to three for most top-ranked clusters.

The results for both cities lie between the simulation's typical and worst scenarios. The results for Boston are closer to the worst scenario, whereas Waltham's are much like the typical. The main reason for this difference is the traffic in Boston that caused more variations in estimating non-idle time than Waltham. The small gap between real and simulation results shows that our simulation framework could serve as an effective model for studying the attack on a larger scale, where experiments are limited. We intend to expand our dataset to include other cities. We also hope that, with our dataset and code made available to researchers, other groups will contribute with measurements from their cities.

Countermeasures

In Android, the location and sensor services are accessed using similar APIs. The main difference between them is the absence of permissions when sensors are accessed. A simple mitigation technique would be to define new permissions for sensor access and force apps to request these permissions from the users. Another difference between the services is the absence of visual notification to users when sensors are accessed. This can be addressed by displaying the sensor name and the accessing app in the notification bar. These solutions should be more carefully investigated in the context of usability and human–computer interaction. If an app genuinely requires sensors to function, complex mitigation techniques are required, such as monitoring Internet traffic for location leakage or monitoring energy consumption. Alternatively, generating adequate artificial noise in the data before providing it to the app can reduce the potential of such attacks.

The techniques discussed here can be integrated in the OS for global protection or implemented using dynamic instrumentation tools (for example, ddi; github.com/crmulliner/ddi) or app sandboxing tools (for example, Boxify[8]).

Our algorithms' performance in both simulations and real experiments indicates that a significant number of users are vulnerable to tracking by seemingly innocuous apps in most cities. This motivates questions regarding the implications of smartphone sensors on users' privacy. Access to sensor information is important for enhanced interaction with the environment, but preventing malicious exploitation and abuse of this information calls for rigorous mitigation methods and tools. ■

References

1. "Android Flashlight App Developer Settles FTC Charges It Deceived Consumers," US Federal Trade Commission, 5 Dec. 2013; www.ftc.gov/news-events/press-releases/2013/12/android-flashlight-app-developer-settles-ftc-charges-it-deceived.
2. D. Patterson et al., "Inferring High-Level Behavior from Low-Level Sensors," *Ubiquitous Computing*, LNCS 2864, A. Dey, A. Schmidt, and J. McCarthy, eds., Springer, 2003, pp. 73–89.
3. S. Narain, A. Sanatinia, and G. Noubir, "Single-Stroke Language-Agnostic Keylogging Using Stereo-Microphones and Domain Specific Machine Learning," *Proc. ACM Conf. Security and Privacy in Wireless & Mobile Networks*, 2014, pp. 201–212.
4. L. Zhang et al., "Senstrack: Energy-Efficient Location Tracking with Smartphone Sensors," *IEEE Sensors J.*, vol. 13, no. 10, 2013, pp. 3775–3784.
5. J. Han et al., "ACComplice: Location Inference Using Accelerometers on Smartphones," *Proc. 4th Int'l Conf. Communication Systems and Networks* (COMSNETS 12), 2012; doi:10.1109/COMSNETS.2012.6151305.
6. S. Nawaz and C. Mascolo, "Mining Users' Significant Driving Routes with Low-Power Sensors," *Proc. 12th ACM Conf. Embedded Network Sensor Systems* (SenSys 14), 2014, pp. 236–250.
7. S. Narain et al., "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors," *Proc. IEEE Symp. Security and Privacy* (SP 16), 2016; doi:10.1109/SP.2016.31.
8. M. Backes et al., "Boxify: Full-Fledged App Sandboxing for Stock Android," *Proc. 24th USENIX Security Symp.* (USENIX Security 15), 2015, pp. 691–706.

Sashank Narain is a PhD candidate in information assurance at Northeastern University. His research is focused on how smartphone sensors can impact users' privacy worldwide and the design and implementation of proof of attacks on smartphones. Narain has an MS in information assurance from Northeastern University. He's a member of IEEE and Information Systems Security Association (ISSA), and cofounded the Northeastern University chapter of ISSA. Contact him at sashank@ccs.neu.edu.

Triet Vo-Huu is a postdoctoral researcher in the College of Computer and Information Science at Northeastern University. His research focuses on user privacy in mobile and cloud applications as well as the security of wireless communications and networks. Vo-Huu received a PhD in computer science from Northeastern University. Contact him at vohuudtr@ccs.neu.edu.

Kenneth Block is a systems/software architect and is pursuing a PhD in information assurance from Northeastern University. His research interests include covert channels and privacy. Block received an MS in

electronic engineering from Northeastern University. He's a member of IEEE and ACM and has a Certified Architect designation from the Open Group. Contact him at block.k@husky.neu.edu.

Guevara Noubir is a professor of computer and information science at Northeastern University. His research focuses on privacy and security. Noubir received a PhD in computer science from Ecole Polytechnique Fédérale de Lausanne. He serves on the editorial board of *ACM Transactions on Privacy Security, IEEE Transactions on Mobile Computing,* and *IEEE Transactions on Information Forensics and Security*. Contact him at noubir@ccs.neu.edu.