Geo-locating Drivers: A Study of Sensitive Data Leakage in Ride-Hailing Services

Qingchuan Zhao*, Chaoshun Zuo*, Giancarlo Pellegrino†‡, Zhiqiang Lin*

*The Ohio State University
†CISPA Helmholtz Center for Information Security
‡Stanford University
{zhao.2708, zuo.118, lin.3021}@osu.edu, gpellegrino@{cispa.saarland, stanford.edu}

***Abstract*—Increasingly, mobile application-based ride-hailing services have become a very popular means of transportation. Due to the handling of business logic, these services also contain a wealth of privacy-sensitive information such as GPS locations, car plates, driver licenses, and payment data. Unlike many of the mobile applications in which there is only one type of users, ride-hailing services face two types of users: riders and drivers. While most of the efforts had focused on the rider's privacy, unfortunately, we notice little has been done to protect drivers. To raise the awareness of the privacy issues with drivers, in this paper we perform the first systematic study of the drivers' sensitive data leakage in ride-hailing services. More specifically, we select 20 popular ride-hailing apps including Uber and Lyft and focus on one particular feature, namely the nearby cars feature. Surprisingly, our experimental results show that large-scale data harvesting of drivers is possible *for all of the ride-hailing services we studied*. In particular, attackers can determine with high-precision the driver's privacy-sensitive information including mostly visited address (e.g., home) and daily driving behaviors. Meanwhile, attackers can also infer sensitive information about the business operations and performances of ride-hailing services such as the number of rides, utilization of cars, and presence on the territory. In addition to presenting the attacks, we also shed light on the countermeasures the service providers could take to protect the driver's sensitive information.**

I. Introduction

Over the last decade, ride-hailing services such as Uber and Lyft have become a popular means of ground transportation for millions of users [34], [33]. A ride-hailing service (RHS) is a platform serving for dispatching ride requests to subscribed drivers, where a rider requests a car via a mobile application (app for short). Riders' requests are forwarded to the closest available drivers who can accept or decline the service request based on the rider's reputation and position.

To operate, RHSes typically collect a considerable amount of sensitive information such as GPS position, car plates, payment data, and other personally identifiable information (PII) of both drivers and riders. The protection of these data is a growing concern in the community especially after the publication of documents describing questionable and unethical behaviors of RHSes [18], [8].

Moreover, a recent attack presented by Pham et al. [30] has shown the severity of the risk of massive sensitive data leakage. This attack could allow shady marketers or angry taxi-cab drivers to obtain drivers' PII by leveraging the fact that the platform shares personal details of the drivers including driver's name and picture, car plate, and phone numbers upon the confirmation of a ride. As a result, attackers could harvest a significant amount of sensitive data by requesting and canceling rides continuously. Accordingly, RHSes have adopted cancellations policy to penalize such behaviors, but recent reported incidents have shown that current countermeasures may not be sufficient to deter attackers (e.g., [15], [5]).

Unfortunately, the above example attack only scratches the tip of the iceberg. In fact, we find that the current situation exposes drivers' privacy and safety to an unprecedented risk, which is much more disconcerting, by presenting 3 attacks that abuse the nearby cars feature of 20 rider apps. In particular, we show that large-scale data harvesting from ride-haling platforms is still possible that allows attackers to determine a driver's home addresses and daily behaviors with high precision. Also, we demonstrate that the harvested data can be used to identify drivers who operate on multiple platforms as well as to learn significant details about an RHS's operation performances. Finally, we show that this is not a problem isolated to just a few RHSes, e.g., Uber and Lyft, but it is a systematic problem affecting all platforms we tested.

In this paper, we also report the existing countermeasures from the tested RHSes. We show that countermeasures such as rate limiting and short-lived identifiers are not sufficient to address our attacks. We also present new vulnerabilities in which social security numbers and other confidential information are shared with riders exist in some of the RHSes we tested. We have made responsible disclosures to the vulnerable RHS providers (received bug bounties from both Uber and Lyft), and are working with them to patch the vulnerabilities at the time of this writing.

Finally, to ease the analysis efforts, we have developed a semi-automated and lightweight web API reverse engineering tool to extract undocumented web APIs and data dependencies from a mobile app. These reversed engineered web APIs are then used to develop the security tests in our analysis.

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23052
www.ndss-symposium.org

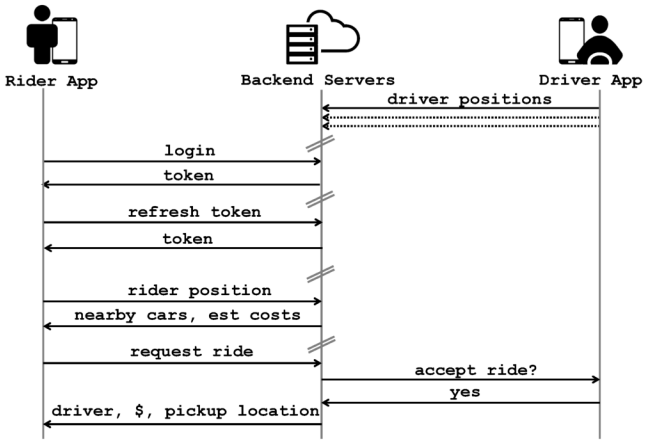


Figure 1: An overview of the web APIs used by RHSes.

```
GET /nearby-cars?lat=33.7114&lng=151.1321
HTTP/1.1
...
```

```
HTTP/1.1 200 OK
Content-type: application/json
...

{
    "cars": [
        {
            "id"    : "509AE827",
            "positions": [
                {
                    "GPS": "-33.7100 / 151.1342",
                    "t"  : "15259620050000"
                }, {
                    "GPS": "-33.7300 / 151.1200",
                    "t"  : "15259620060000"
                },
                ...
        }, {
            "id"    : "6F09E2AA",
            ...
        },
        ...
}
```

Figure 2: An example of a rider request and response message.

Our Contribution. To summarize, this paper makes the following contribution:

- **Novel Attacks** (§V). We present new attacks that are able to extract the privacy sensitive data (some of which can even lead to threat to drivers safety) of ride-hailing drivers. We collect a large volume of data using the nearby cars feature and show that we can determine their home address and other personal behaviors with high precision. We also show that the analysis can reveal information about an RHS's business performances.
- **New Tool** (§III). As the web APIs in our analysis are typically undocumented, we present a novel lightweight dynamic analysis tool to reverse engineer the web APIs and perform our security tests.
- **Empirical Evaluation** (§IV). We present an analysis of Nearby Cars web APIs from 20 RHSes and assess the effectiveness of existing countermeasures.
- **Countermeasures** (§VI). Finally, we also present a list of dos and donts, and discuss more robust countermeasures for protecting driver's privacy in RHSes.

Paper Organization. The rest of the paper is organized as follows. We first provide necessary background in §II and introduce our tool and methodologies for conducting this study in §III. Next, we show the results of our analysis over the web APIs of our interest in §IV, and present three carefully designed attacks in §V. We then discuss our findings and possible countermeasures against our attacks in §VI, and compare with related works in §VII. Finally, we conclude in §VIII.

II. Background

A. About RHSes

Ride-hailing is an emerging mode of ground transportation that a rider can reserve a car service using a mobile app. In general, it works as follows. When the rider inputs a destination address and requests a ride, the mobile app reads the GPS position of the device and transmits it together with the address to the back-end server. Then the server dispatches the request to the available drivers closer to the rider. If an available driver accepts the request, then the server transmits additional information to both the driver (e.g., the pickup location) and the rider (e.g., the estimated time of arrival).

The ride-hailing market is flourishing over the past several years, and many companies have entered this business following the path mapped by Curb, Flywheel, and Uber. Despite the rich variety of offerings, the underlying architectures connecting riders to drivers are very similar to each other. An overview of the most important protocols for such services is shown in Figure 1. In particular, an RHS system is composed of: (i) a mobile app for the rider (rider app), (ii) a cloud of back-end servers, and (iii) a mobile app for the driver (driver app). The rider app is used by customers to request rides. It is connected over the Internet to a cloud of back-end servers that are responsible to authenticate riders (and drivers), and to match riders to drivers. And, the driver app is used by drivers exclusively.

The communication between back-end servers and mobile apps is typically via web APIs—HTTP-based[1] app programming interfaces to execute remote functions. Figure 1 shows five examples of web APIs supporting the basic operations of RHSes:

- **Driver Real-time Position**: The driver app sends a feed of available positions of drivers to the server. The collected positions will later be used by the server to dispatch riders' requests to available drivers;
- **Login**: The Login API is responsible for authenticating users, i.e., both riders and drivers. The mobile app collects username and password of the rider (or the driver), and sends it via an HTTP request to the server. If the authentication succeeds, the server produces an authenticated token that will be used by the mobile app as a mean of authentication proof when sending subsequent HTTP requests.
- **Refresh Token**: Typically, a token can only be used in a limited time window. The Refresh Token API is used to retrieve a new token from the server when the old one expired.

[1] HTTPS protocols are similarly handled with the only difference of either at the networking API interception or with a packet decryption using a controlled proxy.

- **Nearby Cars**: The forth API is used by the rider app to obtain information about nearby cars and a quote of the cost of the ride. Figure 1 shows an example of this API with the request and response message. The request message carries the rider's location and the response message contains several nearby cars. Each car has at least an identifier (`id`), the position information, which includes the GPS coordinates and the time stamp indicating when such position is recorded.
- **Ride Request**: The last API is used to request a ride and spawns across the three entities. It is initiated by the rider when requesting a ride for a specific destination. The server will determine the closest drivers to the rider's current position and ask them if they would accept the ride. If so, the server assigns the first responded driver to the rider, and sends to the rider app the details about the ride.

RHSes may provide additional services and APIs that are not shown in Figure 1, such as billing information for customers and APIs to integrate with other third-party services (e.g., Google Maps).

B. Motivation and Threat Model

Motivation. The motivation of our work is based on a serious attack against drivers of RHSes. To the best of our knowledge, one of the first few attacks threatening the safety of drivers has been presented by Pham et al. [30] as a part of a broader study on privacy threats in RHSes. In this attack, the attacker is a group of angry taxi-cab drivers who wants to harm RHS drivers coordinately. To do so, the attacker exploits the behavior of the Request Ride API that returns drivers' personal details. Based on this behavior, the attacker collects drivers' information by requesting and canceling rides. While this threat may seem implausible, a number of news reports is showing that physical violence is a real threat to RHS drivers (e.g., [39], [10], [21], [31]). On the other hand, RHS providers have begun to charge penalties if users canceling rides. This policy increases the cost for conducting such information collection, and mitigates the attacks utilizing the Request Ride API.

However, despite the Request Ride API, we find that the Nearby Cars API can also leak drivers' information both directly and indirectly. Nevertheless, it remains underestimated and is rarely noticed by attackers and researchers. There might be multiple reasons. The first reason is probably that, showing the nearby cars is a common feature of apps in this category, which brings directly to the users with vivid visual effects and lets them realize how many available cars around them, in order to estimate where they would better to move to catch a car in a shorter time. This feature is provided by almost every RHS app today, though different app may adopt different strategy to display the nearby cars (e.g., using different radius). The second possible reason is that, this API is not designed to provide drivers' information directly as what the Request Ride API does, such as driver's name, plate number, and phone number. As a result, when designing RHS apps, the app developers may intuitively provide this feature by default, without challenging much about its security.

Therefore, in this paper, we intend to systematically study the severity of the data leakages originated from this visual effect, which is brought by the execution of the Nearby Cars API. To our surprise, we find that this feature can actually cause a lot of damages to both the drivers and the platform providers as well.

Service Name	#Downloads	Obfuscated?
Uber	100+ millions	✔
Easy	10+ millions	✔
Gett	10+ millions	✔
Lyft	10+ millions	✔
myTaxi	5+ millions	✔
Taxify	5+ millions	✗
BiTaksi	1+ millions	✔
Heetch	1+ millions	✔
Jeeny	500+ thousands	✔
Flywheel	100+ thousands	✗
GoCatch	100+ thousands	✔
miCab	100+ thousands	✗
RideAustin	100+ thousands	✗
Ztrip	100+ thousands	✔
eCab	50+ thousands	✔
GroundLink	10+ thousands	✗
HelloCabs	10+ thousands	✗
Ride LA	10+ thousands	✗
Bounce	10+ thousands	✗
DC Taxi Rider	5+ thousands	✔

Table I: The selected RHSes in our study.

Threat Model. We assume the attacker is either a ride-hailing service, an individual, or a group of persons. In addition, the attacker can reverse engineer the rider app of RHSes, create fake accounts, use GPS spoofing to forge user positions, and control several machines connecting to the Internet.

III. Methodology and Tools

A key objective of this work is to have a systematic understanding of the current circumstances of driver's security issues in RHSes by studying the related web APIs they exposed. To this end, we intend to investigate the deployed countermeasures or mechanisms that can prevent, increase the cost, or slow down the acquisition of the GPS positions of drivers, and meanwhile to understand whether such data leakage is a threat to drivers' privacy and RHS business. For this purpose, we have to apply security tests over web APIs, which requires proper descriptions of the web API end-points, parameters, and API call sequences. Unfortunately, the documentation of web APIs is not always available: out of the 20 mobile apps we studied, only Lyft provides a description of the Nearby Cars API[2]. To solve this problem, we need to design a tool for web API reverse engineering.

In this section, we first describe how we select the RHSes and their apps in §III-A, then present how we design our web API reverse engineering tool in §III-B and its implementation in §III-C.

A. Selection of the RHSes

We conducted our study on a selection of RHSes by searching for the keyword "ride-hail" on Google Play Store through a clean Chrome Browser instance and selecting the top 20 suggested apps that can be installed and run on our devices

[2]See "Availability - ETA and Nearby Drivers" https://developer.lyft.com/reference

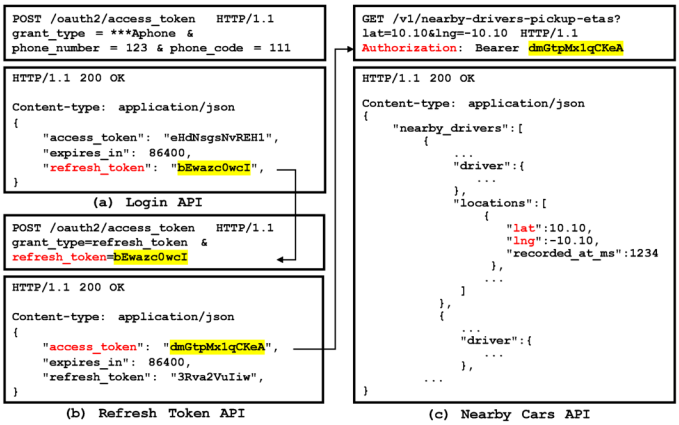


Figure 3: The web APIs and dependencies in Lyft.

on the date of April 3rd, 2018. The selected apps are listed in Table III. Please note that these search engine suggested apps were determined by Google on that particular day, and these apps may not be the top downloaded apps. While we could have just used the top 20 RHS apps based on the number of accumulated downloads in Google Play, the reason of using the suggested RHS apps returned by the search engine is to get a fairly reasonable distribution of these apps.

Then we investigate some general properties of our apps. First, we estimated their popularity by using the number of downloads provided by the Google Play Store, which shows that our apps vary from the world-wide known Uber and Lyft to local services such as RideAustin and RideLA. Further, we looked into details about whether the mobile app has been obfuscated to thwart the app analysis, which is useful for the development of our web API reverse engineering tool. To do so, we manually examined each app's binary code with the help of the tool JEB[3] and found that 12 out 20 apps (60%) have been obfuscated, which makes static analysis of the app challenging. Meanwhile, we also examined the apps' communication security by setting up a man-in-the-middle proxy with customized certificates and we found only Uber enforces the certificate checking.

B. Reverse Engineering of the Web APIs

Running Example. We begin the description of our tool with a running example to illustrate the problems we have to solve for extracting the Nearby Cars API and all of its related APIs. These APIs are required to be executed correctly in a systematic manner to generate our security test results. The running example is from a real app, Lyft.

When opening the Lyft app to use its services, the rider will be asked to provide a phone number to receive a verification code sent from the back-end server via SMS. After providing this verification code, the app invokes the Login API, which is shown in Figure 3(a), where the phone number and the verification code are carried by the parameter `phone_number` and `phone_code`, respectively. It receives the `access_token`, which will be expired in 86,400 seconds as well as the `refresh_token`, which is used later to require a new token when the current one expired.

At the time of a successful login, Lyft triggers the Nearby Cars API automatically. As shown in Figure 3(c), the Nearby Cars API requires three important fields: `lat`, `lng`, and `Authorization`, where `lat` and `lng` represents the user's `geo_location`, and the `Authorization` carries the token value for the authorization purpose. After 86,400 seconds, the old token will be expired, then the app invokes the Refresh Token API as shown in Figure 3(b). This API carries an important parameter, `refresh_token`, whose value comes from the response of the Login API. Next, the invoked Refresh Token API receives the response from the server with a new token, the value of `access_token`, as well as a new refresh token. Later, this new token is carried within the Nearby Cars API for continuously retrieving the data containing nearby cars.

Challenges. From this running example, we can notice a number of challenges we have to solve in order to perform our security analysis.

- **Pinpointing the Web APIs of Interest**. An RHS client app may involve multiple web APIs during the interaction with the servers. For instance, the Uber app actually triggers hundreds of web API calls. We must identify the API of our interest, i.e., the Nearby Cars API. Interestingly, this API does take a parameter with value of GPS coordinates. Identifying such a parameter is helpful to narrow the scope to pinpoint this API.
- **Identifying the Dependencies among APIs**. The parameters of one web API can depend on the values obtained from other APIs. For instance, the value of `access_token` in Nearby Cars API comes from the response of Refresh Token API. Therefore, we also have to identify the closure of the web APIs related to the Nearby Cars API. Obviously we have to perform a dependency analysis of all of the executed web APIs.
- **Bypassing Obfuscations Used in the Apps**. We cannot simply use static analysis to identify the web APIs, because there are 60% of the RHS apps in our dataset that have been obfuscation to thwart our analysis. Meanwhile, as the security analysis involves retrieving nearby cars, the access control token must be provided otherwise the server will reject our requests. Therefore, we have to choose dynamic analysis. In addition, we also cannot simply setup a network proxy to intercept and decrypt the HTTP(S) traffic, because one of the apps (i.e., Uber) performs the certificate checking. Consequently, we have to hook in-app APIs to intercept the network traffic.

Approaches. There are multiple approaches to solve the above challenges. Intuitively, we can use instruction level dynamic taint analysis (e.g., TaintDroid [13]) to understand how the information flows through the app (e.g., how the GPS location and the server response such as token is defined and used by the web APIs) to pinpoint the web APIs of our interest as well as to identify the dependencies. Such a dynamic analysis approach also bypasses static code obfuscation and can intercept the HTTPS traffic at the network API level.

[3]Available at https://www.pnfsoftware.com/jeb/

Interestingly, according to our preliminary analysis of these 20 apps, we also notice that we can use a lightweight API level data dependency analysis instead of the heavyweight instruction level data dependency analysis (i.e., taint analysis) to solve our problem. In that, the parameters are mostly strings and we can identify the dependencies by matching their values. The only limitation for this approach is that we are unable to identify the dependencies if a string is transformed between the definition of the string and the use of the string. Fortunately, we did not notice such a case in our RHS apps.

Therefore, we eventually design a lightweight, API level, dynamic data dependency analysis that works in the following three steps:

Step I: Logging Android and System APIs. First, we instrument a large number of system APIs of our interest, which includes (i) all of the HTTP(S) system libraries (e.g., HttpClient) and low level (SSL)Socket APIs handling third-party or self-developed libraries; (ii) the system APIs that are required by Ride-Hailing services, such as `LocationManager.requestLocationUpdates()`, `LocationManager.getLastKnownLocation()`, `GPSTracker.getLatitude()`, `GPSTracker.getLongitude()`, and `System.currentTimeMillis()`. During the execution of these APIs, we log the name, the parameters, and the return values of the system APIs in a log file.

Step II: Resolving the Web APIs. Unlike the system APIs whose name is documented, we do not have any name of the web APIs because they are merely HTTP request and response messages. On the other hand, these messages have already been logged when the networking system APIs get executed. Therefore, by inspecting the networking request and response API execution information in the log file, we can pair each request with its corresponding response, and then parse these pairs according to the HTTP protocol specification [1]: a request message includes 1) a request-line, 2) request header fields, 3) an empty line, and 4) an optional message-body; and a response message contains 1) a status-line, 2) response header fields, 3) an empty line, and 4) an optional message-body.

Specifically, we parse the request message to obtain the request URL as well as request parameters and we also parse the response messages to abstract its content as a set of pairs of `<field_name,value>`. With respect to the parameters and response value pairs, we parse them accordingly based on their specific encodings (e.g., JSON and XML). Eventually, the web API is resolved by the request URL, the request parameters, and the return values (i.e., response message). Then, we replace the log entires of the original network sending and receiving APIs with the newly resolved web APIs in the log file.

Step III: Data Dependency Analysis. Then by analyzing the log file in both forward and backward directions, we identify the APIs of our interest and also dependencies. In particular:

- **Forward Data Dependency Analysis**. Starting from the return values of the hooked system APIs (e.g., `GPSTracker.getLongitude()`), we search where this value is used in the log file in the forward direction. The web APIs that use the GPS coordinates in the request parameters is the candidate of the Nearby Cars API. Also, interestingly, the GPS coordinates will also be used in the return values of the Nearby Cars API because each nearby car also has a location. An example of this response message is in shown in Figure 2, which is the JSON formatted item in nearby `cars` array. Therefore, to further narrow down the candidate, we also inspect the response messages. If the GPS coordinates exist in the response message, we identify this Nearby Cars API.
- **Backward Data Dependency Analysis**. Having identified the Nearby Cars API, we then search in a backward direction to locate where the parameters of this API are defined. Transitively, we identify the closure that generates the parameters such as the `access_token`. Note that to really identify whether a parameter is token, we apply the same differential packet analysis [2] to infer the tokens in the request message. The key observation is that different users are assigned with different tokens, and we can therefore align and diff their requests for the same web API by using two different users. Such a protocol alignment and diffing approach has been widely used by many protocol reverse engineering systems (e.g., [2], [9], [42], [43]), and we just use the one from the Protocol Informatics (PI) project [2].

C. Implementation

We have implemented our analysis tool atop the Xposed [3] framework, which allows the dynamic interception of all of the Android APIs including system APIs. The execution of these APIs is logged into a log file, in which each entry contains the API name, the value of parameters, and return value. To resolve the web APIs from the log file, we just develop standard parsing with python scripts. In particular, we depend on `urllib`, `zlib`, `json`, and `xml` python libraries to parse and decode the content of the web API. Finally, to infer the tokens in the request and response messages, we use the open source message field alignment and diffing implementation from PI [2].

The last piece of our tool is a standalone data scraping component that is able to collect the nearby driver information by sending a large volume of request messages to the RHS server with proper parameters. With our web API reverse engineering component, the implementation of this task becomes quite simple. In particular, we just developed a python script that sends HTTP(S) request messages to the servers by using the token obtained in the web API reverse engineering and mutating the GPS coordinates of our interest. If the token requires refresh, we execute the refresh token API with proper parameters as well. Please note that these parameters have already been identified by our data dependency algorithm.

To summarize, for each analyzed RHS app, we first installed the app in an instrumented Android device where most of the Android APIs are interposed and their executions are logged. For each selected app, we also created two user accounts for each service. Then, we performed a user login request and reached the view where the cars are displayed on a map, by using the two users we registered. Next, we analyze the log file to resolve the web APIs of our interest and

Rider App	RL1	RL2	SM1	SM2	GPS	AN1	AN2
Uber	●	○	●	∞	○	∞	●
Easy	-	○	○	∞	○	∞	●
Gett	-	○	●	∞	○	∞	●
Lyft	●	○	●	24h	○	∞	○
myTaxi	-	○	○	∞	○	20m	●
Taxify	●	○	●	∞	○	∞	●
BiTaksi	-	○	●	∞	○	∞	●
Heetch	-	○	●	∞	○	∞	●
Jeeny	-	○	○	∞	○	20m	●
Flywheel	-	○	●	20m	○	10m	●
GoCatch	-	○	●	∞	○	∞	●
miCab	-	○	●	∞	○	∞	○
RideAustin	-	○	●	∞	○	∞	●
Ztrip	-	○	●	30m	○	∞	●
eCab	●	○	○	∞	○	∞	●
GroundLink	-	○	○	∞	○	∞	●
HelloCabs	-	○	●	∞	○	∞	○
Ride LA	-	○	○	∞	○	∞	○
Bounce	-	○	●	∞	○	∞	○
DC Taxi Rider	-	○	●	∞	○	∞	○

Table II: List of countermeasures. Values: ● for countermeasure present, ○ for countermeasure missing, "-" for unknown, and ∞ for not expired. Columns: RL1 for Reqs/s, RL2 for Different IPs, SM1 for Authn, SM2 for Session Life-Span, GPS for Anti-GPS Spoofing, AN1 for Identifier Life-Span, AN2 for Driver Info.

identify the dependencies. After that, we run our standalone data scraping component to scrape the nearby cars. We refer to §IV and §V for the description of the individual test of the apps.

IV. Security Analysis of Nearby Cars API

We now present our security analysis of Nearby Cars APIs. The goal of this analysis is to identify server-side mechanisms and possible countermeasures that can block or slow down the attacker's operations. The list of the countermeasures is presented in §IV-A and the analysis results are presented in §IV-B.

A. Analysis Description

The first step of our analysis is to prepare a list of countermeasures to evaluate. We reviewed publicly available documents such as ride-hailing apps' API documentation for developers and the best practices for web service development[4] to search for known countermeasures covering the following categories: rate limiting, anti-GPS spoofing, session management, data anonymization, and anti-data scraping. Table II shows the list of countermeasures. In the rest of this section, we discuss each category and provide details of our tests.

Rate Limiting. Rate limiting is a technique that is used to limit the number of requests processed by online services, and it is often used to counter denial of service (DoS) attacks. Based on our threat model, the attacker can take advantage of multiple computers to perform a large number of requests. Accordingly, we considered two countermeasures: per-user rate limits on the number of requests and per-user limits on the number of IPs used.

(RL1) **Rate Limits Reqs/s**: Servers can limit the number of requests processed over a period of time. The rate limits can be enforced for each user or web server. When the limit is reached, the web server may respond with a "`429 Too Many Requests`" response status. We populated this column using the information we gathered from the ride-hailing service documentations. Only Uber and Lyft describe the rate limits based on the frequency of requests per second and the total amount of requests per user. The other services do not share these details. However, during our experiments, we discovered that Taxify and eCab implement rate limits. Nevertheless, these limits are enforced when administrators suspect undergoing malicious activities, e.g., DoS.

(RL2) **Different IPs**: RHSes may be recording the IPs for every user who logs in as a measure to mitigate session hijacking attacks. When the server detects a new IP, it may require the user to be re-authenticated. To populate this column, we checked the behavior of the server when processing parallel requests from the same user session using different source IPs. We used two sources: an IP of the DigitalOcean Inc. network, and the other of our own campus network.

Session Management. Session management encompasses the mechanisms to establish and maintain a valid user session. It includes user authentication, generation, and validation of session identifiers. In this analysis, we focus on those aspects that can limit attacker activities.

(SM1) **Authentication**: The first aspect we intend to check is whether the access to Nearby Cars API is restricted to the authenticated user only. We verify this by checking for the presence of a session ID in the Nearby Cars API request.

(SM2) **Session Lifespan**: The second aspect is the lifespan of user sessions that may slow down attackers. For example, shorter validity time windows may require the attacker to re-authenticated frequently. We measure the session lifespan by calling the Nearby Cars API over an extended period. When we receive an error message, e.g., HTTP response "`4xx`" series status code or a response with a different response body format (e.g., keys of JSON objects), we mark this session as expired. We did not design ad-hoc experiments for that, but we monitored errors during the experiments of §V.

Anti-GPS Spoofing. The attacker spoofs GPS coordinates to fetch nearby cars. As such, services may deploy mechanisms to verify whether the GPS position is consistent with other measurements, e.g., nearby WiFi networks and nearby cell towers[5]. For this category, we do not enumerate and test possible

[4]See, the "OWASP REST Security Cheat Sheet" https://www.owasp.org/index.php/REST_Security_Cheat_Sheet and the "OWASP Web Service Security Cheat Sheet" https://www.owasp.org/index.php/Web_Service_Security_Cheat_Sheet

[5]See https://developer.android.com/guide/topics/location/strategies

countermeasures, but we verify the presence of mechanisms that would prevent an attacker from rapidly changing position via GPS spoofing. For this test, we spoofed GPS coordinates so that the users will appear in very distant places at the same time. We first identified at least two cities where each ride-hailing service operates. For example, for Lyft, we selected 11 cities and performed one request per second for each city for twenty times. Four services, i.e., Bounce, RideAustin, RideLA, and DC Taxi Rider, operate in a single city. In these cases, we picked distant points within the same city.

Anonymization. This category contains countermeasures to hide sensitive information and make it hard for an attacker to reveal drivers' identities. We derived this list by manually inspecting the content of Nearby Cars API responses.

- **(AN1)** **Identifier Lifespan**: As shown in Figure 2, the Nearby Cars API's responses carry identifiers for either cars or drivers in most cases. In this study, we assume each driver is binding to a unique car, which means the identifier for a car and for a driver is conceptually equivalent. These identifiers can be used to track cars and drivers across different responses. Shortening the lifespan of identifiers may mitigate this problem. Then, we tested the time it takes for an identifier to be updated. As discussed for the session ID lifespan, we measured the identifier lifespans during the experiments of §V.
- **(AN2)** **Personally Identifiable Information**: We inspect the responses looking for personally identifiable information. We looked for the first and last name, email, phone numbers, and others.

B. Results

We now present the main results of our analysis. Results are presented in Table II.

Rate Limiting. Uber, Lyft, and Gett are the only three services provide publicly available API documentations. According to Uber's documentation, Uber enforces a limit of $2,000$ requests per hour and a maximum peaks of 500 requests per second per user. In our experiments, we observed that the real rate limit is much lower, i.e., one request per second. As the Nearby Cars API is undocumented, we speculated that this may be a particular rate limit of the Nearby Cars API only. Lyft reports the presence of rate limits; however, they do not disclose the actual thresholds. Gett does not report the presence of rate limits.

For Taxify and eCab, we discovered rate limits at about two requests per second. These limits were not always present, but they were enforced after they notified us about suspicious traffic originated from our servers.

For the remaining RHSes, we did not identify rate limits. As we elaborate more in §V, we requested on average about four requests per second based on the insight gained with Uber, Taxify, and eCab. Higher rate limits may be present, but we did not verify their presence for ethical reasons. Finally, none of the services enforce a same-origin network policy for user requests.

Service name	Sensitive information
Lyft	Driver avatar
HelloCabs	Name, phone number
Ride LA	Name, phone number
DC Taxi Rider	Name, phone number, email
miCab	Account creating time, account last update time, device number, hiring status
Bounce	Name, date of birth, driver avatar, phone number, social security number, driver license number, driver license expiration date, home address, bank account number, routing number, account balance, vehicle inspection details, vehicle insurance details

Table III: List of personally identifiable information of drivers included in Nearby Cars API responses

User Authentication. 14 services restrict the Nearby Cars API to authenticated users only. The remaining services, i.e., GroundLink, myTaxi, Easy, Jeeny, RideLA, and eCab do not require any form of user authentications. This allows any public attacker to retrieve nearby cars without user authentication.

It is worth to mention the case of GoCatch. Every time a user wants to log in at GoCatch, the service requires the submission of a token sent via SMS. While this approach may affect the service usability, it can raise the cost of the attacker operations.

Session Lifespan. Since the beginning of the experiments, all services—except for three—have not required us to obtain a fresh user session. For Uber, Lyft, Heetch, Gett, and Flywheel, the experiments last in total 28 days. During this period, only Lyft and Flywheel require us to refresh the session ID after 24 hours and every 30 minutes, respectively. For the other services the experiment lasted 15 days (eCab and Taxify only 7 days). Among these, only Ztrip requires to refreshen the session ID every 30 minutes.

Anti-GPS Spoofing. Our analysis did not reveal the presence of any anti-GPS spoofing behavior among all of tested RHSes.

Identifier Lifespan. Overall, 17 services do not use short-lived identifiers. The maximum time interval is the same as that of session lifespan. Only three services shuffle identifiers every 20 minutes. Among these, it is worth mentioning the behavior of Flywheel that refreshes identifiers about every 10 minutes.

Personally Identifiable Information. Our analysis revealed that in total six services share Personally Identifiable Information (PII). Among them, we discovered full names, phone numbers, as well as sensitive information such as social security numbers and bank account data. The complete list of PII per service is in Table III.

C. Takeaway

In short, our first analysis did not observe any particular countermeasures hampering attackers. Instead, our analysis revealed behaviors that can facilitate attackers, e.g., long-lived tokens. Also, our tests identified two types of vulnerabilities in 11 RHSes: six services do not require user authentication to reveal the position of nearby drivers, and other six services directly return a variety of personally identifiable information

Rider App	City/Area	Req/s	Days	Cov/M
Uber	O'ahu Island, Hawai'i	1	28	19
Easy	Sao Paulo, Brazil	4	15	0.3
Gett	Eilat, Israel	4	28	0.3
Lyft	O'ahu Island, Hawai'i	5	28	19
myTaxi	Hamburg, Germany	4	15	20
Taxify	Paris, France	2	7	12
BiTaksi	Istanbul, Turkey	4	15	20
Heetch	Stockholm, Sweden	4	28	12
Jeeny	Riyadh, Saudi Arabia	4	15	0.3
Flywheel	Seattle, US	4	28	7
GoCatch	Sydney, Australia	4	15	20
miCab	Cebu, Philippines	4	15	0.8
RideAustin	Austin, US	4	15	7
Ztrip	Houston, US	4	15	12
eCab	Paris, France	2	7	7
GroundLink	Dallas, US	4	15	20
HelloCabs	Yangon, Myanmar	4	15	7
Ride LA	Los Angeles, US	4	15	20
Bounce	San Diego, US	4	15	20
DC Taxi Rider	Washington DC, US	4	15	3

Table IV: An overview of the parameters of our experiments. Cov/M for the estimate coverage area (mi^2) of one monitor.

(RideLA contains both vulnerabilities), which even includes sensitive and confidential information (e.g., social security numbers and bank account numbers).

V. Attacks

The results of the web API analysis indicate that the Nearby Cars API may be poorly protected. Attackers may be able to collect a large volume of data containing drivers' identifiable information and their positions, which can uncover drivers' sensitive information indirectly. To demonstrate the threats, in this section, we present three attacks to show that the current implementations of Nearby Cars API not only seriously threaten drivers' safety and privacy, but also allow attackers to spy on RHS business performances.

In this section, we present the details of our attacks. First, we present the data collection and processing in §V-A. Then, three attacks are presented in §V-B,§V-C, and §V-D, respectively.

A. Design

Our attacks consist of three components: data acquisition, data aggregation, and data analysis.

Data Acquisition. Data acquisition is performed with *monitors*. A monitor is a bot that controls a rider account. In this study, all monitors for a particular RHS use only one account. A monitor is placed in an appropriate location in a city to collect data by continuously performing API calls with spoofed GPS coordinates and store collected data in a local database. Moreover, monitors are responsible for determining when the authorization token needs to be refreshed.

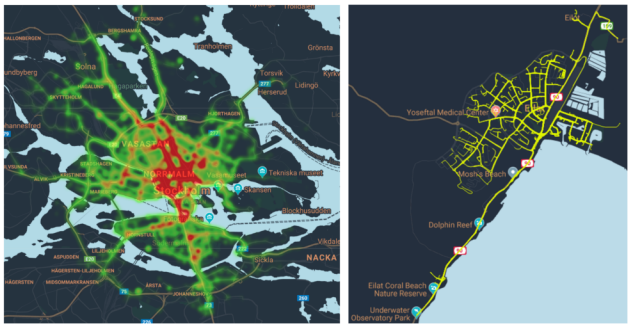
The exact locations of our monitors are determined as follows. First, if the RHS operates in multiple cities, we select a city which is relatively isolated from neighboring cities (e.g., in an island). Second, we calculate the average size that a monitor could cover (up to 20 mi^2 for ethical concerns). Then, we place monitors in a grid based on the size of the area covered by each monitor, which varies considerably across services; however, as cities have irregular shapes, we adjusted monitors to better adapt to the shapes manually. Also, as monitors may cover the same area, we further refined the positions of monitors to reduce overlaps. The locations, coverage size of each monitor, and other parameters of our experiments are reported in Table IV.

After being placed, each monitor starts to acquire data at a constant request rate, which has been determined by considering ethical aspects. Specifically, our experiments must not interfere with the normal business operations of RHSes and not to trigger the active rate-limiting mechanism, if there is any. Accordingly, we first tried to acquire data from Lyft with a rate of 10 requests per second, the documented rate limits. After two hours, we reached the Lyft's rate limit, and we reduced monitors' rate by half, i.e., five requests per second. Then, we used the new rate for Uber. However, we reached the rate limit of Uber as well and further reduced to one request per second. For the other RHSes, we set the initial rate four requests per second and never changed it. Only for Taxify and eCab, we further reduced the request rate to two requests per second.

In fact, we acquired data incrementally. First, we started the acquisition for Lyft, Uber, Heetch, Gett, and Flywheel on April 13th, 2018. The responses data are collected over four consecutive weeks (28 days), i.e., between April 13th and May 10th. Then we extended the acquisition of data to the remaining 15 RHSes from May 11th. In total, except for Taxify and eCab, we acquired data for 15 days. Because of a power outage, our monitors were offline or gathered partial date between May 12th and 14th, and May 19th and 21th. We excluded these days in the following study. For Taxify and eCab, we acquired only seven days because the network providers flagged our machines as infected. Accordingly, we suspended the acquisition of data.

Data Aggregation. Responses of Nearby Cars API return car paths. Each path is a list of timestamped GPS coordinates with an identifier, which is used to link paths to cars or drivers and does not change over time. One of these RHSes, i.e., Lyft, requires additional attention. Lyft's Nearby Cars API responses include the URL of driver's avatar, a driver-chosen picture (selfie in most cases). Avatars do not change very often, and this makes them reliable identifiers for drivers. However, each response contains only the URL of the closest driver. To gather the URLs of other drivers, we deploy a *mobile monitor* for each newly-discovered "driver" to perform an additional API call closer to the most recent GPS coordinate.

Data Analysis. The final step is to remove noises from our dataset. First, we observe that drivers work as *full-time* or *part-time*. We categorize drivers as full-time if they appear more than half of the total number of days. Compared to the part-time drivers, full-time drivers have a tendency to exhibit more regular daily patterns. Thus, we focus on full-time drivers only. Second, drivers have various activities through a day if they are absent in our dataset, giving a ride or logged out of the platform (e.g., to sleep or eat). As none web API we used to collect data can distinguish a specific activity, we rely on the *inter-path interval* to distinguish the two cases. In particular, we observe that the average ride in the cities that we are monitoring could last up to 45 minutes. Accordingly, if the



(a) (b)

Figure 4: (a) Heatmap of Heetch drivers one day in Stockholm; (b) Path of a single Gett driver in Eilat, Israel.

time interval between two consecutive paths is between 5 and 45 minutes, then the driver is treated as giving a ride. Similarly, if the interval is longer than six hours, then the driver is taking a break.

B. Attack #1: Tracking Drivers' Daily Routines

In this attack, we discover that the collected data can be used by attackers to precisely track drivers during their daily routine. First, we show that the information could allow attackers to precisely determine the movements of drivers over time. Then, we demonstrate that the information can also allow attackers to identify drivers' daily behaviors, specifically, their working patterns and the most likely appeared locations with a precision of $100m$.

Movements of Drivers. Figure 4(a) is a heatmap of all Heetch drivers' paths operating in Stockholm in a day, which is drawn by overlapping all paths of Heetch drivers in our dataset. The red color of Figure 4(a) shows the areas where the activities of drivers are more intense, i.e., central Stockholm. The heatmap fades to the green color towards areas less popular, i.e., outskirts of Stockholm. In addition, the collected data allows an attacker to track a single driver too. For example, Figure 4(b) shows all paths of a single Gett driver in Eilat, Israel.

Daily Behaviors of Drivers. Our dataset reveals daily behaviors of drivers. In this attack, we focus on the daily working patterns of drivers, i.e., when to start working, and the most likely appeared locations (e.g., home) over different days at about the same time of a day. Disclosure of drivers' behaviors and locations where a driver mostly visited is a serious sensitive data leakage that threatens drivers' safety. Due to the limitation of computing power, network bandwidth as well as ethical considerations, for some RHSes, our monitors may not cover the entire area of the city. Drivers' behaviors in these uncovered areas may bring noises to our analysis in this attack. For example, if a driver continuously works in the uncovered area, then the related information of this driver is missing from our dataset. In this case, it is possible that this driver is actually working but is considered as taking a break because of being absent from our dataset longer than six hours. Because of these constraints, we eventually choose to only test four RHSes, Uber, Lyft, Gett and miCab, whose monitors cover almost the entire city, for proof of the concept.

RHS	# Total	# Morning	%	# Afternoon	%	# Evening	%
Uber	1,202	705	58.7%	336	30.0%	161	11.3%
Lyft	638	385	60.3%	167	26.2%	86	13.5%
Gett	120	82	68.3%	27	22.5%	11	9.2%
miCab	152	136	89.5%	10	6.6%	6	3.9%

Table V: Daily working patterns of drivers from Uber, Lyft, Gett and miCab.

In addition, to further remove the noise data and simplify our tests, we chose the cities which are either located in an island or relatively isolated from the nearby cities. As an almost closed system, most aspects of the society is expected to remain stable, e.g., the number of cars and drivers, people's life styles. A stable system benefits attackers to retrieve the pattern of these aspects. Specifically, the dataset of Uber and Lyft was acquired in O'ahu Island, Hawai'i, from April 13th, 2018 to May 10th, 2018, the dataset of Gett was acquired in Eilat, Israel, and miCab in Cebu, Philippines from May 11th, 2018 to May 25th, 2018.

Working Patterns: The working patterns of drivers from an RHS is revealed by studying the repetitive behaviors of these drivers. To find out the behaviors, first, we select drivers whose six-hour break is across two consecutive days. Among these drivers, we select those who start working from locations that are within $100m$ from each other. Then, we use the total number of nearby points as a measurement of the precision for detections. By using a low precision of three points, we identified totally $1,202$ Uber drivers, 638 Lyft drivers, 120 Gett drivers and 152 miCab drivers who start working from almost the same location across days. Next, we study the working patterns by classifying them into different work shifts. We separate a day into three shifts, morning (4:00 AM to 12:00 PM), afternoon (12:00 PM to 8:00 PM), and evening (8:00 PM to 4:00 AM next day). If a driver starts working at 9:00 AM, then his or her work shift is in the morning. The result of drivers' daily working patterns is shown in Table V. As we expected, most drivers from any of these RHSes prefer to start working in the morning, less in the afternoon, and the least in the evening.

Appeared Locations — Home: Further analysis of the data may reveal the most likely appeared locations of a driver, if attackers restrict the criteria by increasing the precision of the location detection and narrowing down the time window. Among these locations, we intend to uncover one of the secretest privacy information of a driver — the home address. For this purpose, we focus on drivers who start from the same place between 6:00 AM and 9:00 AM with a probability of 0.5. Our hypothesis is that, if a driver starts working in the morning from the almost the same location which is in a residential area, then such location is mostly like to be his or her home. To validate this hypothesis, we need to plot the GPS coordinates and centroid of the points of such location for each driver on a map and manually verify them. Therefore, we choose to use Uber to verify this hypothesis, because it has the largest number of full-time drivers in our dataset.

As a result, our dataset revealed that 334 Uber drivers start working from the same nearby points for half of the time. Among these 334 drivers, we have identified that 123 of them that start working between 6:00 AM and 9:00 AM. After plotting and manually checking these locations, we verified that 102 of them is located in a residential area, which may suggest that it is nearby the real address in which drivers live; six of them is nearby restaurants; and 15 of them is located nearby gas stations and shopping centers, where may be the places that these drivers are used to having breakfast.

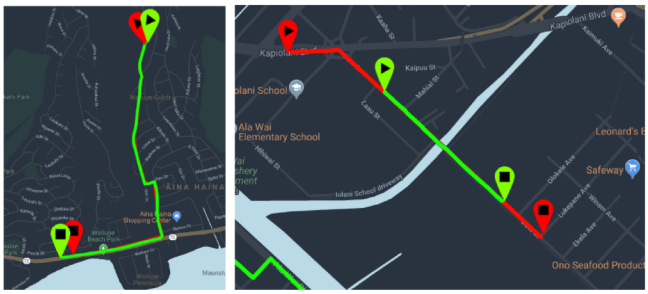
Interestingly, as the data of Uber and Lyft is collected in the same area, we then plotted possible home addresses of their drivers on the map and we discovered a set of overlapping points. Even this overlapping is probably a coincidence, given the observations that many drivers work for both two RHSes, we reasonably question whether our collected data is capable of uncovering driver's employment status? For example, whether a driver only works for one specific RHS or different RHSes at the same time. This inspires us to conduct the following attack, namely attack #2 presented in §V-C.

Takeaway. From this analysis, we showed that the data obtained by Nearby Cars API can reveal the movements of a driver over time. In addition, more seriously, further analysis of these data can also disclose sensitive privacy information, which includes drivers' working patterns and the most likely appeared locations, which could be a restaurant, a gas station, or even the real home.

C. Attack #2: Uncovering Drivers Employment Status and Preference

In addition to the interesting observation about drivers may work for multiple RHSes, this attack is also inspired by a news report. More specifically, Uber was reported to used the *Hell* program to spy on Lyft drivers from 2014 to 2016, in order to identify drivers working for both platforms and convince them to favor Uber with additional financial rewards [11]. But it is still unclear how technically Uber performed such an attack. Therefore, in this attack, we intend to show that it is possible to use our collected data to identify drivers using different platforms simultaneously, and to reveal which platform is more of a driver's favor. We exemplify these attacks on Uber and Lyft whose data is collected on O'ahu Island, Hawai'i, from April 13th, 2018 to May 10th, 2018.

Drivers Employment Status. This attack is to reveal whether a driver is employed by different RHSes simultaneously. The main challenge is to compare data points of Uber and Lyft drivers and look for matches. To reduce the scope of possible matches, we first remove obvious contradictions. For example, drivers that are in two different areas in the same time interval cannot be the same driver. Afterwards, we select all pairs of paths and count the number of points are closer both in space (i.e., 60 meters) and time (i.e., two seconds). In total, we identified 401 drivers that are at the intersection of the 835 Lyft and 1,328 Uber full-time drivers. To validate this results, we randomly selected 100 drivers and plotted their paths on the map for visual verifications. We did not notice any contradiction against the hypothesis that these drivers are working for both platforms.



(a) (b)

Figure 5: Examples of overlapped paths. Green is Uber and Red is Lyft.

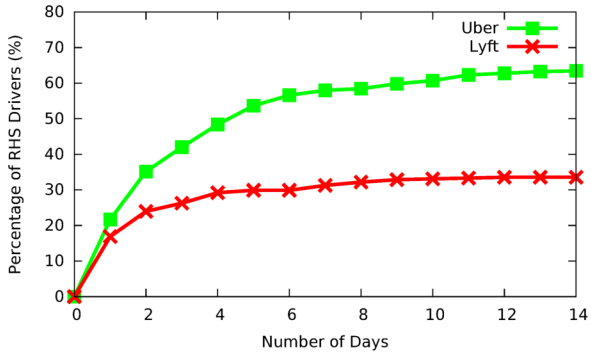


Figure 6: CDF of shared drivers only use Lyft or Uber.

Figure 5 shows two examples of overlapped paths of an overlapped driver. The red path with start and end markers is from the Lyft dataset and the green is from the Uber dataset. In Figure 5(a), the two paths start from almost the same location (start markers are overlapped) but end in different locations; and in Figure 5(b), the red path and the green path starts and ends both in different locations. This could happen, because a driver may not perform the same operations on two apps, for example, closing one app while keeping the other one running.

Drivers Preferences. In addition, our analysis also revealed interesting elements about drivers' preferences: 48% of Lyft drivers is also on the Uber platform, whereas only 30% of Uber drivers is on the Lyft platform. We detailed this aspect by looking at the number of days a driver prefers exclusively working with one RHS or the other. Figure 6 shows the result of this analysis. It indicates that drivers using these two platforms prefer using Uber over Lyft. Because, more than 64% of drivers working for both prefer using exclusively Uber against only 33% drivers prefer Lyft for at most 14 days, i.e., half of the time considered for this analysis.

Takeaway. Overall, our analysis showed that the Nearby Cars API can be used to snoop on drivers using different platforms simultaneously. Interestingly, our analysis showed that drivers operating in O'ahu Island, Hawai'i tend to prefer the Uber platform.

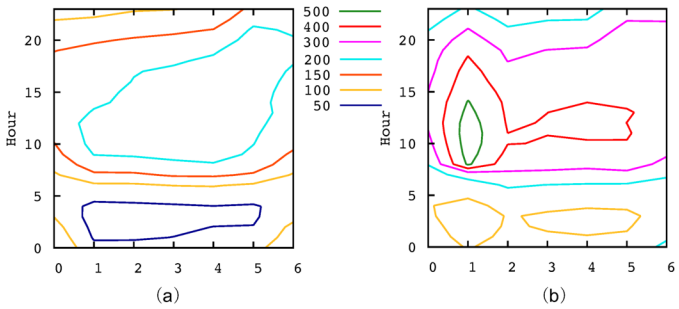


Figure 7: (a) Contour lines of Lyft active drivers; (b) Contour lines of Uber active drivers. On the X axis, 0 is for Sunday and 6 is for Saturday.

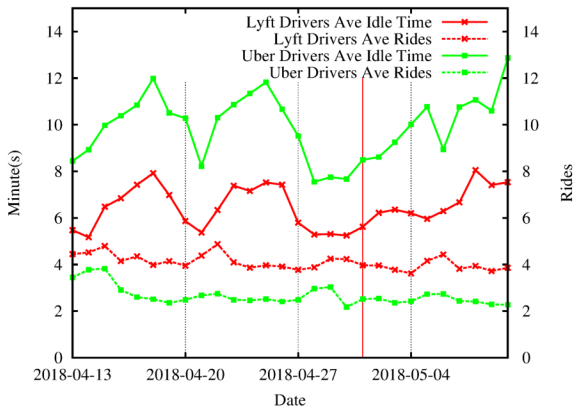


Figure 8: Ave idle time and rides per day of Lyft and Uber

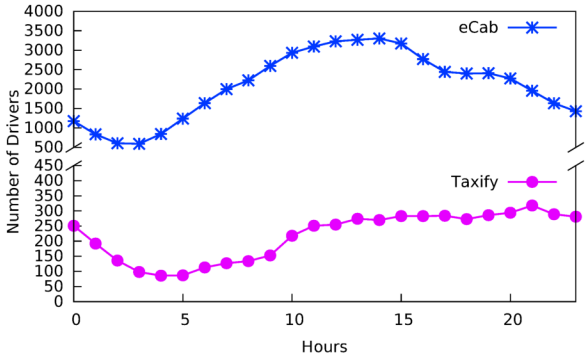


Figure 9: Active drivers in eCab and Taxify.

D. Attack #3: Business Information Leakage

In this attack, we show that the collected data can be used by attackers to extract business information of RHSes. In particular, we focus on RHSes that are operating in the same city, i.e., eCab and Taxify in Paris, Uber and Lyft in O'ahu Island, Hawai'i, though, it can be conducted between RHSes of different areas for espionage as well. Specifically, for each pair of competitors, we extract and compare the metrics and statistics of their operations with each other, which includes the number of drivers, number of rides, distribution of active drivers over weekdays and time of the day, and waiting time. This analysis is not meant to be a complete comparison between organizations; however, it intends to show the feasibility of such an attack.

Lyft vs. Uber. Consider, for example, that one of the two RHSes would like to know the number of cars used by the other competitor as well as the hours and days of activity. The first part of the analysis answers this question by extracting the distribution of number of drivers over weekdays and time of the day from our dataset. Figure 7 shows the average number of drivers for each weekday and hour using contour lines. In the X axis, we use "0" for Sunday and "6" for Saturday. It shows that, from Monday to Friday, drivers from both platforms are more active between 10:00 AM and 5:00 PM, and less active at night from 1:00 AM to 5:00 AM. Over weekends, the activity of drivers shifts to later hours. Second, we observe that, at each given time, Uber has more active drivers than Lyft, i.e., about a factor of $2X$. Also, we notice a peak of Uber drivers on Mondays. We did not observe a similar peak for Lyft. We could not find a reasonable explanation for this observation. Based on this analysis, we can conclude that Uber has a considerable advantage over Lyft on the O'ahu Island.

An RHS may also be interested in comparing the performances of its own operations with its competitors. In practice, RHSes deploy algorithms to match riders to drivers and indicate areas where there is a higher demand of rides. The efficiency and accuracy of these algorithms is crucial to optimize the use of resources. Our dataset can also be used to answer this question. For example, Figure 8 shows the daily average number of rides and average waiting time, which shows periodic patterns over four weeks except for the third week between April 28th and May 2nd. This anomaly is believed to be caused by the Lei Day, a public holidays observed in the State of Hawaii which falls on the 1st of May (vertical orange bar in Figure 8). In addition, Figure 8 also shows that Lyft has a higher average number of rides and a lower average waiting time than Uber, which indicates that Lyft manages to match demand and supply more efficiently than Uber, despite of the lower number of cars.

eCab vs. Taxify. eCab and Taxify are two emerging European organizations. Basically, eCab is an alliance of traditional taxi companies whereas Taxify is a more recent company with a business model similar to Uber and Lyft. These two organizations may have interest in the number of cars owned by the competitor to make decides on the business development. To this end, our analysis discovered $7,973$ cars are operated under eCab, who claims to have $7,700$ cars in Paris[6], and $3,565$ cars are owned by Taxify that claims to operate $2,000$ to $5,000$ cars in Paris [16].

The two organizations may also be interested in extracting the type of clientele of the competitor. Figure 9 shows the distribution of average active drivers from both eCab and Taxify. The number of active drivers increases from 5:00 AM to about 3:00 PM. After that, the number of eCab drivers drops. However, Taxify does not show the same trend. Instead, it keeps a quasi-steady shape till midnight. The type of riders can explain the different evening/night pattern. For example,

[6]See eCab's website https://www.e-cab.com/en/paris/

due to its origins, eCab may have retained most of the riders, e.g., business persons. On the contrary, Taxify may look more attractive for younger riders or tourists.

Takeaway. This attack showed that the Nearby Cars API can be used to extract data to compare performances of competing organizations. For example, we showed that in O'ahu Island, Lyft seems to perform better than Uber: a higher average number of rides and less average waiting time. We speculated that Lyft better matches demand and offer. In Paris, we confirmed that eCab has, overall, a higher number of cars than Taxify. However, our dataset may suggest that Taxify and eCab may have different type of riders.

VI. Discussion

In this section we sum up our findings and discuss possible countermeasures against our attacks.

A. Data Reliability

News reports have pointed out that the cars shown on the map may be fake[7]. However, observations from our results indicate that real drivers are generating the data, supporting Uber's statement that denies the news allegations[8]. First, as presented in §V-D, the identified number of drivers meets reported or officially claimed statistics. Second, the collected data for both Uber and Lyft show an anomaly that could be explained by the celebration of the Lei Day. Despite these observations, we further evaluated the responses of the Nearby Cars APIs to detect instances of fake cars by manually watching the cars in the street. Out of 20 Uber and Lyft cars passing from a given street, all of them were present both on the map and on the street. The average delay between cars appearing on the map and on the street is about five seconds. Then, when a car is shown on the map, the car has no passengers, which indicates that the driver is available for riders. Based on our observations and evaluation, we believe that data shared with the Nearby Cars API is organic and quasi-real-time.

B. Solutions and Pitfalls

Based on the two analyses in §IV and §V, we obtained a list of pitfalls, mitigations, and suggestions in order to solve the security issues presented in this paper.

Rate Limits. In this paper, we showed that a low request rate is sufficient to identify drivers' sensitive privacy. Among 20 services, only three described rate limits in the documentation. However, none of them were sufficient to prevent the attacks presented in §V. In two cases, i.e., Taxify and eCab, even we observed hard rate limits, but these limits were not presented from the very beginning of our analysis. They were introduced after we received the notification of compromisation of our system from the network provider. This suggested that the network providers of Taxify and eCab were supervising network traffics to spot unusual requests to identify compromised machines. In general, we conclude that the rate limiting is not an ineffective countermeasure against the attacks presented in this paper.

[7]See, e.g., http://www.wired.co.uk/article/uber-algorithm-fake and http://www.slate.com/articles/technology/future_tense/2015/07/uber_s_algorithm_and_the_mirage_of_the_marketplace.html

[8]https://www.wired.co.uk/article/uber-cars-always-in-real-time

```
HTTP/1.1 200 OK
Content-type: application/json
...

{
    "cars": [
        {
            //Car 1
            "positions": [
                {
                    "GPS": "-33.7100 / 151.1342",
                    "t"  : "15259620050000"
                }, {
                    "GPS": "-33.7300 / 151.1200",
                    "t"  : "15259620060000"
                },
                ...
        }, {
            //Car 2
            ...
        },
        ...
}
```

Figure 10: An example of Nearby Cars API response without car and driver identifiers.

Concealing Position with Distance. All RHSes that we studied return GPS coordinates of nearby cars. Service providers may consider to conceal the exact locations of cars by returning the distance between drivers and the rider. However, driver's distances could still be used to infer the position of drivers by utilizing distance triangulation. That is, for each car, the attacker needs to perform three requests from three different points to approximate the position of the driver. We consider this to be an ineffective countermeasure.

Linkability. The analysis of collected data points is based on the capability of the attacker to link paths to drivers. In our analysis, 14 services do not directly provide identifiers for drivers, and this is revealed to be an obstacle towards the data aggregation. However, only removing driver identifiers from responses is not a sufficient countermeasure. As we showed, attackers can aggregate whatever identifiers in the response messages over time, which is sufficient for our attacks because these identifiers last long enough to be identified as an equivalent to driver IDs. A stronger countermeasure is to remove any identifiers. For example, the Nearby Cars API response can return a list of grouped timestamped GPS coordinates, one group for each car. An example of such a response is shown in Figure 10 that is derived from Figure 2.

Synthetic Data. Removing identifiers from response messages can partially solve some attacks against driver's privacy in this paper, but the leakage of business information still remains unprotected (e.g., the heatmap of drivers of an RHS). Moreover, we cannot exclude that machine learning expertise can be applied to extract patterns for linking paths to drivers. A possible solution to this threat is to use synthetic data. However, while this may solve the security concerns raised by this paper, riders may notice a mismatch between cars reported by the app and the ones seen on the street that might raise complaints.

Improper Implementation Logic. The Nearby Cars APIs from six RHSes leak personally identifiable information (PII). According to the business logics of RHSes, providing nec-

essary PII to riders is inevitable. However, improper implementation logic may provide the PII to the one who should not receive. For example, a driver's avatar should be provided to the rider who has successfully scheduled a ride, not any other users. Therefore, we consider an appropriate practice is to provide PII after a successful scheduled ride, which can protect drivers' PII from unexpected leakages.

C. Ethical Considerations and Responsible Disclosure

The analysis presented in this paper involved the analysis of remote servers and handling sensitive data of drivers. We addressed the ethics concerns of our study as follows. First, we designed experiments to avoid interfering the normal operations of RHSes. Our experiments (i) used a low request rate, and we adapted it based on the feedback received by the remote servers and (ii) we did not request, cancel or did any other operations that could change drivers behavior. Second, even though the data we collected is accessible to the public and has not been encrypted, our monitors have been implemented to remove sensitive response fields before storing data in our database. In doing so, we are not storing any private data item, such as full names, dates of birth, and social security numbers.

Our analysis identified security issues that need to be addressed by RHSes' developers. We have notified our findings as follows. First, for these RHSes with clear vulnerabilities, e.g., the SSN returned by Bounce and unauthenticated access to the Nearby Cars API, we have followed the notification procedure presented by Stock et al. [35]. After the initial notification, we regularly verify the presence of the vulnerability. If the vulnerability is present, then we send a reminder after two weeks of the initial notification. Second, to adequately address our findings, RHSes developers may need to redesign the web API and the rider app as well. In this case, we have reached out to the developers, and are discussing the details of our findings.

D. Feedbacks After Disclosure

We notified the developers of all 20 RHSes about our results. Eight services shared with us the details of the patch and asked for our feedback. For example, Bounce removed sensitive PII including social security number and bank account number from their response messages, Lyft's Nearby Cars API has stopped providing avatar informations, and Heetch is considering to harden the web API usage by introducing further restrictions such as shorter the lifespan of drivers' IDs. Furthermore, as a result of our notification efforts, Lyft and Uber each awarded us a bug bounty.

E. Lessons Learned

The Unlearned Lesson Despite Media Attention. The massive sensitive data leakage of drivers [30] and the *Hell* program [11] have received extensive media attentions covering both legal and financial impacts. However, despite all these attentions, changes in the platforms, if any, are not perceptible making it possible for an attacker to spy on drivers.

From Security to Safety. Second, most of the attention has been devoted to the industrial espionage between two competitors and a little has been paid to the possible *safety* issues of drivers. Unfortunately, the issues presented in this paper goes beyond the mere computer security issue and touches drivers' safety. As shown in this paper, Nearby Cars APIs can be used to determine driver's home address.

A Market Segment Problem. Finally, a more concerning outcome of our findings is that Uber and Lyft are not two isolated cases. On the contrary, our results show a problem of an entire sector: for all services, it is possible to mount the same set of attacks of inferring driver's home addresses; also, all of these ride-hailing services suffer from at least one vulnerability. Meanwhile, in one case, i.e., Gett, the attacker can directly query a web API to obtain the position of a specific driver, without the need of harvesting API responses.

VII. Related Work

Privacy-Preserving Location-Based Services (LBS). Privacy in LBSes is a long-lasting concern. Many privacy-preserving architecture have been proposed and attempted to address privacy issues in the broader category of LBSes, e.g., location-based Trust for Mobile User-generated Content [23], location-based social networks [17], privacy-preserving location proof updating system [38], privacy-aware location proof architecture [26]. Most recently, Pham et. al. also proposed two privacy preserving LBS systems particularly for ride-hailing services: ORide [29] and PrivateRide [30]. Our work complements these efforts by demonstrating the possible attacks current ride-hailing services still face.

Leakage of Privacy Sensitive Data in Mobile Applications. The detection of data leakage in mobile applications is a challenging problem that has been addressed from different angles using different techniques. For example, Enck et al. [13], Yang et al. [37] and Egele et al. [12] focused on the problem of identifying mobile apps that transmit sensitive data such as GPS position and contact lists without device users awareness. Data leakage can also occur when transmitting user-provided sensitive date. SUPOR [19] and UiRef [4] have been designed to detect these leakages. Finally, data leakage can be the result of exploitations of code vulnerabilities such as code injection vulnerabilities [20] or improper certificate validation [14], or library vulnerabilities [28].

There are also efforts of identifying the privacy leakage of the server response data from mobile apps. For instance, Kock et. al. [22] proposed using both static analysis and dynamic analysis to semi-automatically discover server-based information oversharing vulnerabilities, where privacy sensitive customer information was unexpectedly sent to the mobile apps. Improper implementation of access control mechanism at the server side can also lead to sensitive data leakage from mobile apps, as shown in AuthScope [43] and LeakScope [41]. Our work is inspired by these server side data leakage problems, but we focus on a new context particularly in the ride-hailing service that has not been explored before.

Web API and Protocol Reverse Engineering. To conduct our study, we developed a lightweight dynamic analysis tool to reverse engineer the remote server web APIs for privacy sensitive data analysis. In fact, there is also a large body of

research focusing on reverse engineering of network protocols from both network traces and application binary executions. In particular, Discoverer [9] and Protocol Informatics [2] extract protocol format from the collected network traces, whereas Polyglot [7], AutoFormat [24], Dispatcher [6], Reformat [36] instead extract protocol format based on how network message is processed by the application binary. Inferring the protocol format is not the primary goal of our analysis. Recently, WARDroid [27] introduces a static-analysis based method to extract web APIs, but it focuses on the implementation logic, which is not the objective of our analysis. However, our technique can certainly integrate these techniques to recognize the message format in addition to the discovery of the web APIs.

Dynamic Analysis of Mobile Apps. Our approach is based on dynamic analysis to identify web APIs and dependencies. Similarly, dynamic approaches have been used in the past to study specific security problems. For instance, TaintDroid [13] has been used to detect whether user's privacy sensitive information can be leaked outside the phone; AppsPlayground [32] recognizes the user interfaces of mobile apps and generates corresponding inputs to expose more app behaviors; DECAF [25] navigates various activities of mobile apps to discover potential Ads flaws; SmartGen [40] executes a mobile app with selective concolic execution to expose malicious URLs; so on and so forth.

Our approach differs from these existing techniques as follows. First, we solve the problem of extracting web APIs including the parameter roles from mobile apps. Second, each work has their own unique challenges. For instance, we do not face the issues of executing all the possible program paths of a mobile app, and instead we rely on security analysts to execute the app. Certainly, we can integrate existing efforts such as SmartGen [40] to expose the web APIs more efficiently and automated.

VIII. Conclusion

We have presented a large-scale study of the privacy-sensitive data leakage of drivers in the ride-hailing services. We focus on one particular feature, namely the nearby cars feature, which retrieves nearby car's information from the server when a rider opens the mobile app. Surprisingly, our study with 20 ride-hailing services including both Uber and Lyft has revealed that the data harvesting attacks are feasible. In particular, our study showed that these attacks are a real threat to the safety of drivers: attackers can determine the locations of drivers with high-precision, including but not limited to the home address, and detect driver's daily behaviors. Moreover, some of the services also leak other confidential information such as the social security numbers of drivers. Furthermore, the aggregated business information about the ride-hailing services can also be learned by attacks such as the number of rides, utilization of cars, and presence on the territory. In addition to evaluating the current countermeasures and reporting the attacks we conducted, we have also discussed more robust countermeasures the service providers could use to defeat the attacks presented in this paper.

Acknowledgment

We would like to thank our shepherd Nick Nikiforakis and the anonymous reviewers for their very helpful feedbacks. This research was supported in part by AFOSR under grant FA9550-14-1-0119, NSF awards 1834213, 1834215, and 1834216, and the German Federal Ministry of Education and Research (BMBF) through funding for the CISPA-Stanford Center for Cybersecurity (FKZ:13N1S0762). Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily of the AFOSR, BMBF, and NSF.

References

[1] "Hypertext transfer protocol," https://www.w3.org/Protocols/rfc2616/rfc2616.html.

[2] "The Protocol Informatics Project," *http://www.baselineresearch.net/PI/*.

[3] "Xposed module repository," http://repo.xposed.info/.

[4] B. Andow, A. Acharya, D. Li, W. Enck, K. Singh, and T. Xie, "Uiref: Analysis of sensitive user inputs in android applications," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '17. New York, NY, USA: ACM, 2017, pp. 23–34. [Online]. Available: http://doi.acm.org/10.1145/3098243.3098247

[5] BBC News, "Uber sues Indian rival Ola over 'fake accounts'," https://www.bbc.com/news/business-35888352, March 2016, [Online; accessed 08-May-2018].

[6] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering," in *Proceedings of the 16th ACM Conference on Computer and and Communications Security (CCS'09)*, Chicago, Illinois, USA, 2009, pp. 621–634.

[7] J. Caballero and D. Song, "Polyglot: Automatic extraction of protocol format using dynamic binary analysis," in *Proceedings of the 14th ACM Conference on Computer and and Communications Security (CCS'07)*, Alexandria, Virginia, USA, 2007, pp. 317–329.

[8] J. Constine, "Former employees say Lyft staffers spied on passengers," https://techcrunch.com/2018/01/25/lyft-god-view/, January 2018, [Online; accessed 07-May-2018].

[9] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *Proceedings of the 16th USENIX Security Symposium (Security'07)*, Boston, MA, August 2007.

[10] R. Dillet, "Protesting Taxi Drivers Attack Uber Car Near Paris," https://www.fastcompany.com/3024798/angry-taxi-drivers-attack-uber-cars-in-paris, January 2014, [Online; accessed 08-May-2018].

[11] A. Efrati, "Uber's Top Secret "Hell" Program Exploited Lyft's Vulnerability," https://www.theinformation.com/articles/ubers-top-secret-hell-program-exploited-lyfts-vulnerability, April 2017, [Online; accessed 07-May-2018].

[12] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS : Detecting privacy leaks in iOS applications," in *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, 6-9 February 2011, San Diego, CA, USA*, San Diego, UNITED STATES, 02 2011. [Online]. Available: http://www.eurecom.fr/publication/3282

[13] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 393–407. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[14] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in)security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[15] E. Fink, "Uber's dirty tricks quantified: Rival counts 5,560 canceled rides," http://money.cnn.com/2014/08/11/technology/uber-fake-ride-requests-lyft/index.html, August 2014, [Online; accessed 08-May-2018].

[16] S. Ghosh, "Taxify has launched in Paris after being kicked out of London," http://www.businessinsider.com/taxify-launched-paris-kicked-out-of-london-2017-10, Oct 2017, [Online; accessed 07-May-2018].

[17] W. He, X. Liu, and M. Ren, "Location cheating: A security challenge to location-based social network services," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 740–749.

[18] K. Hill, "God View': Uber Allegedly Stalked Users For Party-Goers' Viewing Pleasure (Updated)," https://www.forbes.com/sites/kashmirhill/2014/10/03/god-view-uber-allegedly-stalked-users-for-party-goers-viewing-pleasure, October 2014, [Online; accessed 07-May-2018].

[19] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "Supor: Precise and scalable sensitive user input detection for android apps." in *USENIX Security Symposium*, 2015, pp. 977–992.

[20] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, "Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 66–77. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660275

[21] Keep Talking Greece, "Angry taxi drivers on strike attack Uber Taxis in downtown Athens (videos)," http://www.keeptalkinggreece.com/2018/03/06/uber-taxi-attacks-strike/, March 2018, [Online; accessed 08-May-2018].

[22] W. Koch, A. Chaabane, M. Egele, W. Robertson, and E. Kirda, "Semi-automated discovery of server-based information oversharing vulnerabilities in android applications," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2017, pp. 147–157.

[23] V. Lenders, E. Koukoumidis, P. Zhang, and M. Martonosi, "Location-based trust for mobile user-generated content: Applications, challenges and implementations," in *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '08. New York, NY, USA: ACM, 2008, pp. 60–64. [Online]. Available: http://doi.acm.org/10.1145/1411759.1411775

[24] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, San Diego, CA, February 2008.

[25] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: Detecting and characterizing ad fraud in mobile apps," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 57–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616455

[26] W. Luo and U. Hengartner, "Veriplace: A privacy-aware location proof architecture," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '10. New York, NY, USA: ACM, 2010, pp. 23–32. [Online]. Available: http://doi.acm.org/10.1145/1869790.1869797

[27] A. Mendoza and G. Gu, "Mobile application web api reconnaissance: Web-to-mobile inconsistencies and vulnerabilities," in *Proceedings of the 39th IEEE Symposium on Security and Privacy (SP'18)*, May 2018.

[28] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A large-scale study of mobile web app security," in *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.

[29] A. Pham, I. Dacosta, G. Endignoux, J. R. T. Pastoriza, K. Huguenin, and J.-P. Hubaux, "Oride: A privacy-preserving yet accountable ride-hailing service," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1235–1252. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/pham

[30] A. Pham, I. Dacosta, B. Jacot-Guillarmod, K. Huguenin, T. Hajar, F. Tramèr, V. D. Gligor, and J. Hubaux, "Privateride: A privacy-enhanced ride-hailing service," *PoPETs*, vol. 2017, no. 2, pp. 38–56, 2017. [Online]. Available: https://doi.org/10.1515/popets-2017-0015

[31] L. Prinsloo, "South Africa Meter-Taxi Operators Attacking Uber Drivers," https://www.bloomberg.com/news/articles/2017-07-17/south-africa-meter-taxi-operators-attacking-uber-drivers, July 2017, [Online; accessed 08-May-2018].

[32] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 209–220. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435379

[33] SimilarWeb, "SimilarWeb - Traffic Overview of Lyft.com," https://www.similarweb.com/website/lyft.com, 2018, [Online; accessed 07-May-2018].

[34] ——, "SimilarWeb - Traffic Overview of Uber.com," https://www.similarweb.com/website/uber.com, 2018, [Online; accessed 07-May-2018].

[35] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes, "Hey, you have a problem: On the feasibility of large-scale web vulnerability notification," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 1015–1032. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/stock

[36] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "Reformat: Automatic reverse engineering of encrypted messages," in *Proceedings of 14th European Symposium on Research in Computer Security (ESORICS'09)*. Saint Malo, France: LNCS, September 2009.

[37] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS'13)*, November 2013.

[38] Z. Zhu and G. Cao, "Applaus: A privacy-preserving location proof updating system for location-based services," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 1889–1897.

[39] G. Zoroya and A. Waters, "Uber under assault around the world as taxi drivers fight back," https://www.usatoday.com/story/news/world/2015/07/07/uber-protests-global-germany-france-taxi/29500747/, July 2015, [Online; accessed 08-May-2018].

[40] C. Zuo and Z. Lin, "Exposing server urls of mobile apps with selective symbolic execution," in *Proceedings of the 26th World Wide Web Conference (WWW'17)*, Perth, Australia, April 2017.

[41] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, San Francisco, CA, May 2019.

[42] C. Zuo, W. Wang, R. Wang, and Z. Lin, "Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'16)*, San Diego, CA, February 2016.

[43] C. Zuo, Q. Zhao, and Z. Lin, "Authscope: Towards automatic discovery of vulnerable authorizations in online services," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS'17)*, Dallas, TX, November 2017.