A Study of the Privacy of COVID-19 Contact Tracing Apps

Haohuang Wen, Qingchuan Zhao, Zhiqiang Lin, Dong Xuan, Ness Shroff

Department of Computer Science and Engineering
The Ohio State University
{wen.423, zhao.2708, lin.3021, xuan.3, shroff.11}@osu.edu

Abstract

The COVID-19 pandemic has spread across the globe and resulted in substantial loss of lives and livelihoods. To effectively fight this pandemic, many digital contact tracing mobile apps have been developed. Unfortunately, many of these apps lack transparency and thus escalate concerns about their security and privacy. In this paper, we seek to perform a systematic and cross-platform study of the privacy issues in official contact tracing apps worldwide. To this end, we have collected 41 released apps in total, many of which run on both iOS and Android platforms, and analyzed both their documentation and binary code. Our results show that some apps expose identifiable information that can enable fingerprinting of apps and tracking of specific users that raise security and privacy concerns. Further, some apps have inconsistent data collection behaviors across different mobile platforms even though they are designed for the same purpose.

1 Introduction

The COVID-19 pandemic has rapidly spread across more than 180 countries around the world and the death toll has passed 700,000 [6] within only a few months after it first appeared in December 2019. Though governments and healthcare authorities around the world have quickly responded to this pandemic, at the time of this writing, only a few countries have successfully brought it under control. From their practices, *contact tracing* is a widely recognized and effective strategy to monitor and control the spread of the virus.

Contact tracing is an infectious disease control strategy that aims at identifying people who may have come into contact with an infected individual. It is not a new technique and has been performed in past pandemics including SARS in 2003 and H1N1 in 2009. Conventionally, contact tracing is conducted manually starting from collecting the necessary information from infected patients, such as locations they visited and people they had met, via an extensive interview. Unfortunately, manual contact tracing can result in inaccuracies because of the unreliable human memory as well as contacts with strangers. Further, manual contact tracing can also result in large delays, which could reduce its effectiveness.

In order to overcome the above problems in manual tracing, numerous digital contact tracing systems have been recently developed using camera footage, or credit card transaction, or information in cellular network (e.g., cellular tower) or smartphones. Among these, using information exchanged and stored in the smartphone for automated contact tracing is considered one of the most promising solutions for at least two reasons. First, smartphones are ubiquitous. Today, there are 3.5 billion smartphone users (1/3 of the entire population) worldwide, and more than 77% of Americans have smartphones [1]. Second, smartphones have many sensors and communication channels. For instance, a smartphone can easily acquire location information from GPS sensors, communicate with the Internet through the cellular network or Wi-Fi, and exchange information among themselves directly using Bluetooth.

While smartphone-based digital contact tracing is quite promising, it has raised privacy concerns since it can easily become a surveillance system if not properly designed and implemented. To operate, contact tracing apps require their devices to collect a large amount of privacy information (*e.g.*, identifiable information of users). However, such data collection process lacks transparency. That is, according to our observation, only a few official contact tracing apps have been open-sourced and the majority of them remain close-sourced as of June 2020. Though many of these apps have announced to apply privacy-preserving protocols, these publicly disclosed protocols have also been criticized as sacrificing privacy in certain levels. For example, even with Bluetooth Low Energy (BLE) in which no real location is used, it could have various data leakages [19] [17], and meanwhile the identity of an infected user could be de-anonymized by authorities or other users [16].

Motivated by such lack of transparency as well as the privacy concerns, we would like to conduct a systematic study of the official mobile contact tracing apps that have been released by governments and healthcare authorities. While there are many aspects to consider, we focus on the following ones: (*i*) which type of privacy-related information (*i.e.*, information that reveals one's identity) has been collected for contact tracing? (*ii*) are these apps designed and implemented correctly to avoid privacy leakage? (*iii*) is the data being transmitted to other parties, *e.g.*, servers or other users, privacy preserving (*e.g.*, has the data been protected against eavesdropping, tracking, and de-anonymization attacks)? and (*iv*) do these apps behave consistently in different mobile platform?

To this end, we have collected a set of 41 official mobile contact tracing apps on both Android and iOS platforms (26 unique ones in total) as of June 15, 2020 and used a set of techniques to analyze the data privacy of the contact tracing apps. Specifically, for each app, we first recognize contact tracing relevant APIs to detect whether the app uses GPS or BLE to track users. Next, we de-compile the app and analyze the code to identify the privacy-related data collected for contact tracing, including those for BLE broadcasting and those collected along with the GPS data. Finally, we cross-compare the results from the same app between Android and iOS to investigate its behavior discrepancy.

Contribution. In summary, in this paper, we make the following contributions:

- **Systematic Study.** We conduct the first cross-platform study on the user privacy of 41 official contact tracing mobile apps, by analyzing their binary code to reveal the type of privacy-related data collected for the contact tracing purpose.
- **Novel Discovery.** We have uncovered that many of the apps can be fingerprinted with static UUIDs, and two apps are vulnerable to user tracking due to their fixed user IDs.
- **Cross-platform Comparison.** We compared the behaviors of an official app in both the iOS version and the Android version, and we found discrepancies in two apps across the two platforms.

2 Background

2.1 Digital Contact Tracing

Contact tracing has been used by public health authorities to monitor and control the spread of infectious diseases for a long time [24]. Being essentially a tracking system, it has to collect relevant data (*e.g.*, location) to track the contact. Unlike most location-based tracking, such as map navigation, that have to pinpoint users to a specific physical area, *e.g.*, a specific building with the exact physical location labelled by the latitude and longitude, not all digital techniques used in contact tracing request for such precise data, because the goal of contact tracing is just to understand whether two persons are in close proximity. Accordingly, we can classify the types of data used for tracing into the following two categories.

(I) Data for Location Tracing. Collecting the exact locations a person has visited and linking these locations together with timestamps is the most straightforward way to track users (*e.g.*, geo-locating drivers [44]). Meanwhile, a smartphone can provide a variety of data, by either reading from the hardware layer, *e.g.*, GPS sensors, or software layer, *e.g.*, information about a Wi-Fi hotspot, that can be used to pinpoint the specific location of a user. Based on how a user is tracked, we can further break the data for location tracing into two subcategories:

- **Continuous Coordinates-based Data.** A location can be recognized by its GPS coordinates. To obtain such data, we can directly read the coordinates of a smartphone from its embedded GPS sensor. In addition, we can also use cell tower and Wi-Fi to estimate an approximate location within an area. Moreover, even IP addresses can also be used to guess locations with coarse precision, *e.g.*, the street address.
- **Discrete Places-based Data.** In addition to tracking users using the continuous coordinates, we can also profile the movements of a user based on discrete places. For example, we can know where a user has visited by using fixed surveillance cameras, requiring users to check-in in certain places via QR code, or even collecting the transaction histories of credit cards that contain location information. By collecting discrete places, we can uncover the places users have visited.

(II) Data for Proximity Tracing. Unlike conventional location tracing, proximity tracing, theoretically, only measures the distance between two encounters without requiring any identifiable location information in any precision level. In addition, it requires data exchanges between users. In this case, Bluetooth Low Energy (BLE) is a well-suited technology because its signal strength can be used to estimate distances as well as its low energy consumption [21].

2.2 BLE in Proximity Tracing

The BLE, short for Bluetooth Low Energy, is a wireless communication technology with considerably lower energy consumption. In recent years, it has been widely deployed in wearable, smart homes, beacons, and car dongles [39]. Its features are well-suited for mobile contact tracing for three reasons. (*i*) BLE only consumes a low amount of energy to keep its normal operations; (*ii*) It can satisfy the requirement of proximity tracing as only a small amount of data is needed to be exchanged by design; (*iii*) The strength of BLE signal power can be used to calculate the distance between two contacts, which is a required functionality of proximity tracing. Additionally, in BLE communication, there are two important components worthy of mentioning, namely GATT and UUID.

- **GATT**. The Generic Attribute Profile (GATT) is the foundation of communication between two BLE devices that defines the procedures and format of data for transmission. GATT has a hierarchical structure with two key attributes: *service* and *characteristic*. In particular, each service represents a specific property and contains a number of characteristics, and each characteristic stores a piece of data of such property. Additionally, a characteristic can include several descriptors to provide its detailed information.
- **UUID**. The universally unique identifier (UUID) is an important component in BLE communication. It is a hexadecimal string used to uniquely represent an attribute, *e.g.*, service, characteristic, and descriptor. In addition, a UUID could be either a SIG-approved one or a vendor-specific one. That is, in theory, each UUID should be globally unique and the Bluetooth SIG has provided a set of standard UUIDs. For example, according to Apple/Google's Notification Exposure protocol, the service UUID `0xFD6F` has been assigned by Bluetooth SIG for contact tracing [10]. But in practice, SIG also allows different manufactures to use their customized UUIDs, that must be different from the standard ones, for specific purposes.

Workflow of BLE Communication. The workflow of BLE communication involves three primary procedures: (*i*) Broadcasting and Connection, (*ii*) Pairing and Bonding, and (*iii*) Communication.

- **Broadcasting and Connection.** A connection is established between a BLE central device and a BLE peripheral device. To initiate a connection, the peripheral device needs to broadcast its advertisement packets to declare its existence. Whenever a central device notices such a peripheral device, it can actively establish a connection based on the data carried within the advertisement packets, *e.g.*, UUIDs.

- **Pairing and Bonding.** When a connection is established, the central and peripheral devices need to create a channel for secure communication. The processes to establish such a channel are pairing and bonding. In particular, the pairing process works by exchanging supported pairing protocol as well as negotiating a long term key (LTK) for data encryption. Then the bonding process will ask the two devices to store such LTK and use this key to encrypt the data transmitted through the established channel.
- **Communication.** After pairing and bonding, now two paired devices can communicate with each other by exchanging data whose format follows the GATT hierarchy. In particular, the central device first obtains the list of services and characteristics from either the advertisement packets or directly asking from the peripheral device. Next, with the list of characteristics, the central device can operate on values stored in a characteristic, *e.g.*, read and write, if holding sufficient permissions.

Typically, there are two ways to achieve contact tracing using BLE. In the first approach, a smartphone (when acting as a central) directly uses the received broadcasting packets sent from a peripheral (another smartphone), and records the received random cryptographic IDs parsed from the packets, without really establishing any connection. As such, each smartphone will also work as a BLE beacon, which is a 1-way transmitter that keeps broadcasting its identifiers to nearby BLE-enabled devices. The second approach to achieve contact tracing requires device mutual connection, as what has been done in BlueTrace [5] protocol. To be more specific, the two devices first discover each other based on a static UUID from the broadcast packet, and then establish a connection. Next, they exchange the contact information (*e.g.*, a user ID) by writing and reading from a specific characteristic in turn.

2.3 Centralized vs. Decentralized Mobile Contact Tracing

Depending on where the contact detection is performed, there are two typical architectures among current mobile contact tracing systems: (*i*) centralized in which all the detection is performed at a central server, and (*ii*) decentralized in which each client (*i.e.*, the smartphone) performs the detection. To do so, each user's (including the diagnosed positive patient) encounter record is uploaded to a central trusted server periodically in the centralized architecture, and then the central trusted server performs the contact tracing and notifies those who have been in contact with the patient. However, in the decentralized architecture, only the diagnosed positive patient's record is uploaded to the server, and each smartphone will periodically pull the record from the server and then perform the detection locally. There are still heated debates on which architecture is better, though the trend is moving towards the decentralized one especially given the decentralized industry standard set by Apple and Google [7].

3 Methodology

3.1 Scope and Overview

The goal of this work is to analyze COVID-19 contact tracing mobile apps with a focus on the user privacy (*e.g.*, whether there is any privacy leakage or over collecting of user data) from the program analysis perspective. In the following, we define the scope of our analysis, including the privacy issue and program code of our interest, followed by the overview of our analysis.

Privacy of our interest. While there is a broad range of privacy issues that potentially exist in mobile apps, not all of them are of our interest. In fact, many of the them (*e.g.*, information leakage, over-granted permissions) have been well-studied in the literature [41] [42]. Consequently, in this work, we particularly focus on the user privacy issues that are resulted from the misuse of the data being collected for contact tracing purpose. For instance, what type of user data is collected? Can such data reveal user identity?

Program code of our interest. Since our analysis is performed on both Android and iOS apps, the program codes of interest are disassembled or decompiled Java bytecode, Objective-C code, or Swift code.

Overview of analysis. Our analysis can be broken down into three phases. In particular, given an Android or iOS COVID-19 app, we first decompile it and recognize the APIs involved in contact tracing (§3.2). Next, based on these APIs, we identify the privacy information collected for contact tracing (§3.3). Finally, we perform a cross-platform comparison analysis of the corresponding apps to further investigate the discrepancies of the same app (§3.4).

3.2 Contact Tracing Relevant API Recognition

The first step for our analysis is to identify the source information collected for contact tracing in a given app. As described earlier in §2.1, there are two types of sources for contact tracing: cryptographic tokens exchanged through BLE channel, or GPS coordinates acquired from smartphone sensors. Fortunately, all of these operations have to pass through system-defined APIs provided by the mobile operating systems. Therefore, recognizing these APIs will enable the identification of information sources of our interest. To this end, our analysis first decompiles the app binary, which is achieved through the off-the-shelf tools including ApkTool [3] and IDAPro [12]. Next, from the decompiled code, we run a simple script to scan and detect the APIs defined in Table 1, including all the APIs in Android and iOS for BLE and GPS. If any API in the BLE or GPS category is invoked, it implies that very likely the app has used BLE or GPS for contact tracing.

Platform	**Tracking Source**	**API Name**
Android	GPS	Location: float getAccuracy
		Location: float getAltitude
		Location: double getLatitude
		Location: double getLongitude
		Location: float getSpeed
		LocationManager: void getCurrentLocation
		LocationManager: Location getLastKnownLocation
		LocationManager: void requestLocationUpdates
		LocationManager: void requestSingleUpdate
	BLE	BluetoothLeAdvertiser: void startAdvertising
		BluetoothLeAdvertiser: void startAdvertisingSet
iOS	GPS	void CLLocationManager.requestLocation
		void CLLocationManager.startUpdatingLocation
	BLE	void CBPeripheralManager.startAdvertising

Table 1: Relevant APIs for contact tracing in apps.

3.3 Privacy Information Identification

Given the recognized APIs, we further identify the privacy information collected for contact tracing. More specifically, we need to analyze where they are defined, and how they are used, *etc.*, in order to recognize the privacy issues. However, since a mobile contact tracing app can use either GPS or BLE to track users, the collected data can be different regarding different techniques. As a result, we correspondingly have two different approaches to identify them. In the following, we first present the approach of how we recognize BLE-specific data, and then how we recognize location tracing related data.

(I) Identifying BLE-specific Data. Given the nature of the BLE protocol discussed in §2.2, there are two ways for smartphones to exchange contact information: one is through broadcasting of BLE packets (*e.g.*, BLE beacon [21]), and the other is through reading characteristic values via an established BLE connection. As a result, according to these two ways, we have summarized all the related system APIs for Android and iOS in Table 2. For instance, the `startAdvertising` API begins broadcasting of BLE packets, and the `setValue` API sets the value of a BLE characteristic. Given these APIs, we first locate them in the decompiled app code through a searching script by using their names. Next, having identified where these APIs get invoked, we resolve the parameter values since not all of them are directly hardcoded. In particular, since there are quite a number of advertising configurations and data carried by these APIs, we only resolve the parameters that may lead to privacy concerns, such as the advertised data in characteristics and advertisement configurations (*e.g.*, whether the device name is included).

To this end, we have developed a tool to collect the program traces using a backward program slicing algorithm [38], which is based on Soot [13] for Android

Platform	API Name
Android	AdvertiseSettings: AdvertiseSettings.Builder setAdvertiseMode
	AdvertiseSettings: AdvertiseSettings.Builder setConnectable
	AdvertiseSettings: AdvertiseSettings.Builder setTimeout
	AdvertiseSettings: AdvertiseSettings.Builder setTxPowerLevel
	AdvertiseData: AdvertiseData.Builder addManufacturerData
	AdvertiseData: AdvertiseData.Builder addServiceData
	AdvertiseData: AdvertiseData.Builder addServiceUuid
	AdvertiseData: AdvertiseData.Builder setIncludeDeviceName
	AdvertiseData: AdvertiseData.Builder setIncludeTxPowerLevel
	BluetoothGattCharacteristic: BluetoothGattCharacteristic
	BluetoothGattCharacteristic: boolean setValue
iOS	void CBPeripheralManager.startAdvertising
	CBMutableCharacteristic CBMutableCharacteristic.init
	CBMutableService CBMutableService.init
	void CBPeripheral.writeValue
	CLBeaconRegion CLBeaconRegion.initWithUUID

Table 2: Targeted APIs for private data collection for BLE.

and IDAPro [12] for iOS. After collecting all necessary program traces, we need to understand the parameter (*i.e.*, their semantics). While there exists systematic approaches to infer semantics of parameters such as ClueFinder [30] that leverages program elements (*e.g.*, variable names), these approaches require a significant amount of data to train their machine learning models. In our case, due to the small number of apps we have, we applied a manual approach instead, in which we extract the data semantics based on the semantic clues (*e.g.*, variable types, names, logs, and API documentations).

BLE configuration identification. Based on the APIs for `AdvertiseSettings` in Android and `startAdvertising` in iOS, we are able to obtain the following configurations from the function parameters. Note that in iOS, a broadcasting peripheral device can only configure the device name and service UUID while others are controlled by the system by default [15].

– **(C1) Broadcasting Timeout**: When an app turns the phone into peripheral mode and broadcasts packets, it can set a timeout limit for broadcasting. By default, there is no timeout limit for broadcasting.
– **(C2) Device Connectable**: When turning the phone into a BLE peripheral, not all apps would allow other devices to connect for further communication. By default, this value is set to be connectable, which implies that other devices can connect to it and access (*i.e.*, read and write) the BLE characteristics.
– **(C3) Device Name**: The device name can be involved in the advertised BLE packet, and is by default the device name defined in the OS.
– **(C4) TxPower**: This power value, carried in the advertised packet, is often used in BLE proximity tracing for calculating the distance between

two users. By default, this is set to be a medium power strength defined by the OS.

Private BLE data identification. In BLE, the privacy information (*e.g.*, user identifier) can be stored in the manufacture, service, and characteristic data. As such, we leverage the `AdvertiseData` and `setValue` APIs in `Android`, as well as the `init` APIs in iOS to extract these data. Note that we also need to manually infer the semantics of the extracted data, by using binary code level information such as variable types and names.

- **(P1) Manufacture Data**: The broadcasting packet can carry manufacture data along with the manufacture ID, and the manufacture data can be customized that might contain private information.
- **(P2) Service Data**: Similar to the manufacture data, this value is often customized by each app, and thus it may also carry privacy information.
- **(P3) Characteristic Data**: The value stored in each characteristic is for data exchange among devices, which might be privacy-related. For example, two smartphones may exchange the contact information by reading the user identifier from a characteristic.

Device fingerprintable data identification. Previous studies [20] [45] have demonstrated that several attributes in BLE can be used for device fingerprinting. We thus focus on the following fingerprintable BLE data, which can be identified from the `BluetoothGattCharacteristic`, `addServiceUUID`, `init`, and `initWithUUID` APIs.

- **(F1) Manufacture ID**: The manufacture ID, or company identifier, is uniquely assigned by the Bluetooth SIG [4] for each member. Therefore, it can be potentially used for fingerprinting the device manufacture.
- **(F2) Service UUID**: Since each UUID serves as a unique identifier for a service or characteristic, it can be used to fingerprint a BLE device if it remains static. For instance, a nearby user may know from the broadcast UUID that someone is using a certain contact tracing app. However, in BLE proximity tracing, UUIDs could be dynamic values that iterate overtime acting as an anonymity of a user.
- **(F3) Characteristic UUID**: Similar to service UUID, a static characteristic UUID can also enable fingerprinting attacks and thus is also of our interest.

(II) Identifying Location Tracking Related Data. Unlike the data collection through BLE proximity tracing, data collected for GPS location tracing needs to be first stored in the local device (*e.g.*, in a database), and then submitted to the central service when the user is tested positive for COVID-19. Therefore, we currently focus on the related APIs for database operations, which are listed in Table 3 to identify these data. In addition, if a piece of privacy data (*e.g.*, system version and device name) is collected and written into a database

Platform	API Name
Android	void: SQLiteDatabase execSQL long: insert long: insertOrThrow long: insertWithOnConflict
iOS	int sqlite3_finalize int sqlite3_prepare int sqlite3_prepare_V2 int sqlite3_execute

Table 3: Targeted APIs for private data collection stored in database.

along with the GPS data, we speculate that such data will be sent to the server ultimately.

As shown in Table 3, the database APIs, such as `executeSQL`, take a SQL statement as input to execute it. Therefore, we infer the data semantics from the SQL statements that create a table (*e.g.*, `CREATE TABLE table_name (column type, ...)`), where the metadata of the table can be understood by an analyst.

3.4 Cross-platform Comparison

Due to the fact that iOS and Android are the two most dominant mobile operating systems, official contact tracing apps often provide both versions to attract more users. While these apps are released by the same government or healthcare authority, each pair of apps is supposed to behave in the same way, which has been revealed in previous works on other types of apps [40]. However, it is still an open question of whether there are any discrepancies among contact tracing apps across different platforms. Therefore, in the final step of our analysis, we conduct a cross-platform comparison to understand and assess the behavior discrepancies. Specifically, for each pair of apps available in both iOS and Android, we manually compare every perspective of data revealed in the previous phases. For instance, we compare the semantics of the data under the same characteristic (identified with the same UUID) between the two platforms, and observe if there is discrepancy between the app's behaviour.

4 Evaluation

We have applied our methodology to a set of COVID-19 contact tracing apps. In this section, we first describe how we select these apps in §4.1, and then present the identified privacy information collected by these apps in §4.2, and finally show the results of our cross-platform study in §4.3.

4.1 COVID-19 Mobile App Collection

Today, there are numerous COVID-19 themed mobile apps. In this work, we focus exclusively on mobile contact tracing apps. Other types of COVID-19 themed apps (*e.g.*, self-diagnosis apps and treatment guidance apps) are out of scope of this study. In addition, apps under our study should been released and deployed by governments or healthcare authorities on Google Play and Apple App Store. Therefore, we exclude apps that are still under development or are built for demos, concept proving, or other purposes.

Since there is no centralized repository for COVID-19 contact tracing apps, we have to search through the Internet and app store to know which government or authority has released or planned to roll out an official contact tracing app. Fortunately, there are many efforts that have been made by different groups of researchers to maintain a list of such apps, though there is no single list that has covered all apps. Therefore, we built our dataset by combining the apps from these open lists [11] [8] with our own efforts.

In total, we have built a dataset of 41 apps, including 26 Android apps from Google Play and 15 iOS apps from Apple App Store, as of June 15, 2020. Except one app (MyTrace) that exists only in Android, there are 25 apps available in both platforms. However, not all the 25 iOS apps could be downloaded because some of them (*e.g.*, Stopp Corona) have restricted the location for downloading and we were not able to obtain them from our location. Additionally, for the iOS apps, we had to use a jail-broken iPhone to download them and extract the app code from the device. Next, we introduce these 41 apps in the following dimensions:

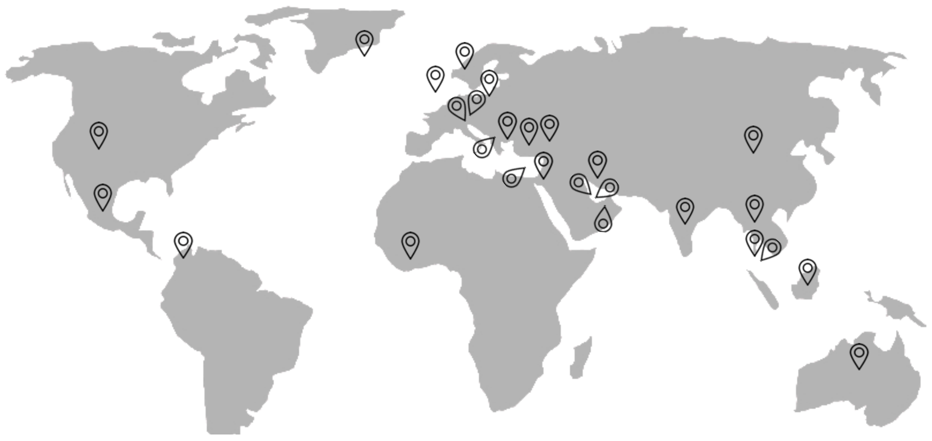

Fig. 1: Distribution of contact tracing apps in our study.

- **Distribution.** The list of all 41 contact tracing apps is shown in Table 4, and we plot their distribution on the map in Figure 1. As shown, we can notice that the contact tracing apps are deployed across six continents and most of them are located in Europe, followed by Asia and then North America.
- **Platform.** According to the third column of Table 4, it is indicated that there is one app developed by Malaysia that supports only Android, while the rest supports both Android and iOS.

App	**Country**	**Plat.**	**Tech.**	**Arch.**	**M1**	**M2**	**M3**
COVIDSafe	Australia	Both	P	C	✔	✔	✔
Stopp Corona	Austria	Both	P	D	✔	✔	✔
BeAware	Bahrain	Both	P; L	-	-	✔	-
ViruSafe	Bulgaria	Both	L	C	-	✔	✗
Chinese health code system	China	Both	L*	C	-	-	-
CoronApp	Colombia	Both	P	-	-	-	-
CovTracer	Cyprus	Both	L	-	✔	-	✔
eRouska	Czech	Both	P	-	✔	✔	✔
StopCovid	Georgia	Both	P; L	C	-	-	-
GH COVID-19 Tracker	Ghana	Both	L	-	-	-	-
Rakning C-19	Iceland	Both	L	-	✔	✔	✔
Aarogya Setu	India	Both	P; L	C	✗	✔	✗
PeduliLindungi	Indonesia	Both	P	-	-	✔	-
Mask.ir	Iran	Both	L	-	-	-	-
HaMagen	Israel	Both	L	D	✔	✔	✔
MyTrace	Malaysia	Android	P	D	-	-	-
CovidRadar	Mexico	Both	P	C	✗	-	✗
StopKorona	North Macedonia	Both	P	D	-	✔	✔
Smittestopp	Norway	Both	P; L	C	✗	✔	✔
ProteGO	Poland	Both	P	D	✔	-	✔
Ehteraz	Qatar	Both	P; L	C	✗	-	✗
Trace Together	Singapore	Both	P	C	✔	✔	✔
MorChana	Thailand	Both	P; L	-	-	✔	✔
Hayat Eve Sigar	Turkey	Both	P; L	C	✔	✗	✗
TraceCovid	UAE(Abu Dhabi)	Both	P	D	✔	-	-
NHS COVID-19 App	UK	Both	P	C	✔	✗	-
Healthy Together	UTAH(USA)	Both	P; L	C	-	✔	✔

Table 4: Contact tracing apps in our study and their meta information (**M1**: claimed minimized data collection, **M2**: claimed limited data usage, **M3**: claimed data destruction). C: Centralized, D: Decentralized, P: Proximity, L: Location, L*: location with QR code.

– **Technique.** As indicated in the forth column, among the 26 unique apps in both platforms, there are 20 apps that adopt BLE proximity tracing, 14 apps that use location tracing, and 8 apps that use both for the purpose of accuracy improvement.
– **Architecture.** Based on our best effort, we can identify the corresponding architectures of 18 apps. In particular, 12 apps are implemented using a centralized architecture, while the remaining 6 use a decentralized architecture. This implies that most governments tend to use a centralized server to collect contact tracing data as of this writing.
– **Description.** We also studied the official description of the apps [11]. Specifically, we checked whether the developer has claimed minimized data collection, limited data usage, and data destruction [11], and the results are correspondingly shown in the last three columns. We find that over half of the apps have declared at least one of these claims, which indicate many of these apps are aware of preserving user privacy.

4.2 Evaluation Result

BLE-Specific Data. In total, 20 apps use BLE for contact tracing. We successfully collected all of them from the Android platform but could only downloaded 10 of them from the iOS platform due to location restriction. The collected BLE-specific data can be classified into two categories: broadcasting data sent to all nearby devices, and BLE property data that can be read when the smartphone is being connected. We present the results in terms of their types in the following.

– **Broadcasting data.** In Table 5, we present the measurement results of the broadcasting parameters, corresponding to the data types defined in §3.3. According to the tenth column of the table, there are 10 apps that use static UUIDs (*i.e.*, UUIDs that do not change overtime) for broadcasting. We further extract the UUID values and summarize them in Table 6. We can notice that these UUIDs are highly customized across different apps (*i.e.*, none of the app shares the same UUID with others). As a result, by reading such broadcast UUIDs, one is able to fingerprint the specific contact tracing app, which to some extent compromises user privacy. In addition to these 10 apps, we also find that 6 apps use dynamic UUIDs that rotate periodically. These UUIDs are generated randomly or through some advanced cryptographic algorithms, which implies that their developers may be aware of such fingerprinting attacks. To summarize, we have the following key finding:

  Finding 1: 10 apps broadcast customized and static UUIDs that enable contact tracing app fingerprinting.

  Further, we discover that the Aarogya Setu app from India has explicitly included the device name (*e.g.*, Alice's phone) in the broadcast packet, and interestingly, this practice only exists in the Android version. As in the app code, it invokes the `setDeviceName` API to set the device name with the value of `unique_id` of a user. Using such device name can raise privacy concern, because it can serve as the unique identifier to track a user.
– **BLE Property data.** In Table 6, we present the extracted BLE property data with their semantics. Note that not all the BLE apps have configured such property data, since some apps only configure information in the broadcasting packet (*e.g.*, use UUID as a user identifier). Interestingly, in addition to the user identifier, we also discover that many apps have collected other device information. For instance, as indicated in the table, 4 apps collect the smartphone model, and 1 app collects device OS version, which are set as characteristic values for nearby devices to read. We further investigated their official documentations, and found that these information serve as factors to calculate the proximity distance between devices, which increases the estimation precision [5] [43]
  Among these results, our key finding is that two apps including Aarogya Setu from India and Hayat Eve Sigar from Turkey directly store fixed user identifier in readable characteristics. Specifically, Aarogya Setu first queries a unique ID from the server and stores it locally. When the `startAdvertising` API

App	Country	C1	C2	C3	C4	P1	P2	F1	F2
COVIDSafe	Australia	0	✔	0	3	✔	✗	✔	Static
Stop Corona	Austria	0	✔	-	3	✗	✔	✗	Dynamic
BeAware	Bahrain	0	✔	-/-	2	-	✗	✔	Dynamic
CoronApp	Colombia	0	✔	0/1	3	✔	✗	✔	Static
eRouska	Czech	0	✗	0/0	2	✗	✗	✗	Static
Aarogya Setu	India	0	✔	1/0	0	✗	✗	✗	Static
StopKorona	North Macedonia	0	✗	-/1	3	✗	✔	✗	Static
MyTrace	Malaysia	0	✔	1	1	✗	✗	✗	Dynamic
CovidRadar	Mexico	0	✔	-/0	0	✗	✗	✗	Dynamic
Smittestopp	Norway	0	✔	0	2	✗	✗	✗	Static
ProteGO	Poland	0	✔	-/1	2	✗	✗	✗	Dynamic
Ehteraz	Qatar	0	✗	0/0	2	✗	✗	✗	Dynamic
Trace Together	Singapore	0	✔	0/1	3	✗	✗	✗	Static
MorChana	Thailand	0	✔	-	2	✗	✔	✗	Static
Hayat Eve Sigar	Turkey	0	✔	0	1	✗	✗	✗	Static
NHS COVID-19 App	UK	0	✔	1/1	2	✗	✗	✗	Static

Table 5: Evaluation result of BLE broadcasting (results separated by / are respectively for Android (left) and iOS (right)).

is called, the app retrieves the ID, and sets it as both the advertised device name and also the value of a readable characteristic. As for the other app, Hayat Eve Sigar also stores the current user's ID in a readable characteristic. As these property data are not protected and can be read once the smartphone is being connected, such fixed IDs can be obtained by nearby users, which can lead to tracking of a specific user. For instance, if the fixed ID appears again in the same location, or different location, an attacker is able to link these locations and IDs to a specific person. This finding has been disclosed to these two apps' developers.

Finding 2: Two apps store fixed user identifiers in their readable characteristics, which allows tracking of a specific user.

Location Tracking Related Data. Among the 26 unique contact tracing apps, 14 of them have used GPS for tracking. The detailed results of the related data collected and stored into database are shown in Table 7. We then manually categorized the data into 6 types, including ID (*e.g.*, user ID), system version (*e.g.*, Android 7.0), device model (*e.g.*, Samsung Galaxy S6), orientation (*e.g.*, landscape and portrait), UI information (*e.g.*, UI style and brightness), and build number. Similar to the device data collected for BLE, we speculate that these data are also collected to improve the accuracy of distance measurements.

Finding 3: It is surprising that contact tracing apps often collect other device information (*e.g.*, system version, and phone model).

App Name	Type	UUID	Semantics	Property
COVIDSafe	S	Random	Monitoring Service	
	C	`B82AB3FC-1595-4F6A-80F0-FE094CC218F9`	ID, model, version, RSSI	R; W
CoronApp	S	`92959161-C063-4613-8AF8-9191408DD389`	Monitoring Service	
	C	`76FE5EB0-F79B-4CE0-8481-59044968DF04`	ID, model, version, RSSI	R; W
eRouska	S	`1440DD68-67E4-11EA-BC55-0242AC130003`		
	C	`9472FBDE-04FF-4FFF-BE1C-B9D3287E8F28`	Current ID	R
Aarogya Setu	S	`45ED2B0C-50F9-4D2D-9DDC-C21BA2C0F825`		
	C	`8D75EA37-6482-4EF5-9FFE-A5E4F44CBEE5`	Unique ID	R; N
	C	`91567DDF-9A75-4FE7-A0AB-F83F4DE15E2F`	PinggerValue	R; N
	C	`5CA2B7AE-EB74-46F4-B161-3C0A6F17F3EC`	Device OS	
StopKorona	S	`0000FF01-0000-1000-8000-00805F9B34FB`		
Smittestopp	S	`E45C1747-A0A4-44AB-8C06-A956DF58D93A`		
	C	`64B81E3C-D60C-4F08-8396-9351B04F7591`		R
ProteGO	C	Random	ID, model, version, RSSI	R; W
Trace Together	S	`B82AB3FC-1595-4F6A-80F0-FE094CC218F9`		
	C	`117BDD58-57CE-4E7A-8E87-7CCCDDA2A804`	ID, model, version, RSSI	R; W
MorChana	S	`000086E0-0000-1000-8000-00805F9B34FB`		
Hayat Eve Sigar	S	`D28ABA6E-EB1F-4193-8CFF-9EDEA7F9E57F`		
	C	`98023D4C-DAE7-4D4E-92C5-2800AFC4512E`	Exchange Message	
	C	`3A8E1D5C-F472-4D41-B33B-C7018CFBAE02`	User ID	R
NHS COVID-19 App	S	`C1F5983C-FA94-4AC8-8E2E-BB86D6DE9B21`		
	C	`D802C645-5C7B-40DD-985A-9FBEE05FE85C`	Keep alive	R; W; N
	C	`85BF337C-5B64-48EB-A5F7-A9FED135C972`	Identity	R

Table 6: Evaluation result of BLE properties (S: Service, C: Characteristic, R: Read, W: Write, N: Notify).

4.3 Evaluation Result of Cross-platform Comparison

Based on the previous results, we further perform a cross-platform study on the apps available in both platforms. We observe that most of the apps have consistent behaviour, which means they collect the same types of user data on both platforms. Interestingly, we also observe some discrepancies in two apps.

The two apps are Aarogya Setu from India and eRouska from Czech. In particular, the Android version of Aarogya Setu involves a unique ID and a Boolean value in two characteristics, while its iOS version have three characteristics instead: a device ID, device OS, and a field called `PinggerValue`. Obviously, the iOS version exposes the device OS information while the Android version does not. Regarding the app eRouska, its Android version does not specify any characteristic, while its iOS version sets a periodically changed user ID in a readable characteristic. It is interesting to observe these discrepancies, as the same pair of app in different platforms is supposed to behave consistently. We suspect this is due to different development processes between the two platforms.

5 Discussion

5.1 Limitations

While we have performed a cross-platform study on the COVID-19 contact tracing apps, there are still several limitations which need additional work to address.

App	ID	SysVer.	Model	Orientation	UI Info.	Build
BeAware Bahrain	✔	✔				
CovTracer	✔	✔	✔			✔
eRouska	✔			✔		
StopCovid			✔	✔	✔	
GH COVID-19 Tracker		✔	✔	✔	✔	✔
Rakning C-19		✔	✔		✔	✔
Aarogya Setu		✔				
HaMagen	✔	✔	✔	✔	✔	
CovidRadar.mx	✔	✔	✔			
StopKorona	✔	✔	✔	✔	✔	
ProteGO	✔	✔		✔		
Trace Together		✔		✔		
NHS COVID-19 App		✔	✔			
CoronApp	✔	✔	✔			

Table 7: Device information collected in apps that use GPS tracing.

First, the set of apps we collected is incomplete, especially the iOS apps, which is due to the restrictions from the downloading regions. In addition, given the fact that the COVID-19 pandemic continues to spread across the globe, many countries will also likely roll out new contact tracing apps for mitigation. Therefore, it is also important to vet these newly emerged apps to ensure user privacy. Secondly, while our focus has been on privacy aspects of contact tracing apps, there are still open research questions that need to be answered. For example, how secure are these apps, are these apps efficient, *etc.*

5.2 Mitigation on the Privacy Issues Identified

Ensuring anonymity. Our findings uncover that some of the apps are vulnerable to fingerprinting and user identity tracking. The root cause is that they use static information such as UUIDs and user IDs that do not change periodically. To mitigate the fingerprinting attack, all contact tracing apps can use a unified UUID for broadcasting. For instance, the `0xFD6F` UUID [10] has been reserved by the Bluetooth SIG for contact tracing purpose. To mitigate user tracking with fixed IDs, developers should integrate dynamically rotatable user IDs, as demonstrated in many of the apps such as eRouska.

Improving transparency. Our study also uncovers discrepancies between Android and iOS apps. Meanwhile, we find that most devices have collected excessive device information (*e.g.*, device OS and model), whose purposes remain unknown. Additionally, as of June 25th, 2020, only a handful of apps are open-sourced (*e.g.*, NHSX [14], Aarogya Setu [2], and eRouska [9]). Overall, the transparency of these contact tracing apps can still be improved, and many technical details related to user privacy should also be open to public. For instance, the usage of the collected private information, and the algorithm and factor used for distance estimation.

6 Related Work

COVID-19 contact tracing app analysis. Since the emerging of COVID-19 contact tracing apps, many efforts have been devoted to studying the security and privacy of these apps, given the nature that these apps need to collect meaningful identifiable user information. There have been a handful of works summarizing existing contact tracing apps and protocols in the world as well as putting forward the current challenges and research questions. For example, Tang [35] analyzed the existing solutions such as MPC [31] and DP-3T [36], and uncovered drawbacks in terms of precision, authenticity, and transparency. Sun et al. [34] seek to vet the security and privacy of contact tracing apps with a focus on only Android apps. Other works [28] [33] [22] also mention some of the potential challenges and convey concerns regarding user privacy.

There are also a few works focusing on single contact tracing app or protocol. For instance, one of the earliest released contact tracing apps, TraceTogether has been studied by a number of researchers (e.g., [22] [28] [26]). In addition, there are also other targets that have been studied, including the NHSX app from UK [37], the contact tracing protocol proposed by Google and Apple [18] [27]. Compared to these existing efforts, our work is the first cross-platform study on both the Android and iOS contact tracing apps with a focus on user privacy from a program analysis perspective.

Proximity tracing with BLE and its security. Prior to contact tracing, BLE has been widely used for geo-localization. In particular, as an energy-efficient wireless technology, BLE has been adopted by beacon devices to enable indoor positioning [21] [25], where the Received Signal Strength Indicator (RSSI) is measured to estimate the distance between two BLE devices [29]. This technique is later adapted in the contact tracing setting to detect whether two people have been in close contact. However, there are also a few attacks on the BLE protocol and devices, such as eavesdropping attacks [32], identity tracking [23], and more recently device fingerprinting [20] [45]. Our analysis with COVID-19 apps also reveals the fingerprinting weaknesses, which may compromise user privacy.

7 Conclusion

In this paper, we present the first cross-platform study of the COVID-19 contact tracing apps, with a focus on user privacy. Starting from the program analysis perspective, we design a methodology to recognize contact tracing relevant APIs, identify the private information collected by the apps, and finally perform a cross-platform comparison on the apps available in both Android and iOS. We have applied our methodology to 41 contact tracing apps (26 Android apps and 15 iOS apps), in which we have obtained a number of privacy concerning findings: one specific app uses default device name for broadcasting which can be used to fingerprint specific users; Two apps store user's fixed IDs in characteristics, which essentially allows user tracking; There are discrepancies across platforms for two apps. Our future work includes improving our analysis to make it more automated, vetting the new emerging apps, and inspecting other issues such as the security of the COVID-19 apps.

Acknowledgement

We would like to thank our shepherd Adwait Nadkarni, and the anonymous reviewers for their valuable feedback. We are also grateful to Juanru Li, Tielei Wang, and Zhi Zhou for their assistance on obtaining the most recent iOS COVID-19 apps. This work was supported in part by the National Science Foundation (NSF) under Grant No. CNS 1834215 and CNS 2028547. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. 60+ revealing statistics about smartphone usage in 2020. `https://techjury.net/blog/smartphone-usage-statistics/`. (Accessed on 06/24/2020).
2. Aarogyasetu_android. `https://github.com/nic-delhi/AarogyaSetu_Android`. (Accessed on 06/23/2020).
3. Apktool. `https://ibotpeaches.github.io/Apktool/`. (Accessed on 06/23/2020).
4. Bluetooth sig, inc. `https://www.bluetooth.com/`.
5. Bluetrace protocol. `https://bluetrace.io`. (Accessed on 06/23/2020).
6. Covid-19 map. `https://coronavirus.jhu.edu/map.html`. (Accessed on 08/06/2020).
7. Covid-19 tracing apps now live in germany, france, and italy; u.k. rethinks its plans. `https://venturebeat.com/2020/06/19/covid-19-tracing-apps-now-live-in-germany-france-and-italy-u-k-rethinks-its-plans/`. (Accessed on 06/24/2020).
8. covid19-tracker-apps. `https://github.com/fs0c131y/covid19-tracker-apps`. (Accessed on 06/23/2020).
9. erouska-android. `https://github.com/covid19cz/erouska-android`. (Accessed on 06/23/2020).
10. Exposure notification - bluetooth specification v1.1. `https://www.blog.google/documents/62/Exposure_Notification_-_Bluetooth_Specification_v1.1.pdf`. (Accessed on 06/24/2020).
11. A flood of coronavirus apps are tracking us. now it's time to keep track of them. `https://www.technologyreview.com/2020/05/07/1000961/launching-mittr-covid-tracing-tracker`. (Accessed on 06/23/2020).
12. Ida pro - hex rays. `https://www.hex-rays.com/products/ida/`. (Accessed on 06/23/2020).
13. Instrumenting android apps with soot. `https://github.com/soot-oss/soot/wiki/Instrumenting-Android-Apps-with-Soot`. (Accessed on 06/23/2020).
14. Nhsx. `https://github.com/nhsx`. (Accessed on 06/23/2020).
15. startadvertising(_:) | ios developer documentation. `https://developer.apple.com/documentation/corebluetooth/cbperipheralmanager/1393252-startadvertising`. (Accessed on 06/23/2020).

16. Bahrain, kuwait and norway contact tracing apps among most dangerous for privacy. https://www.amnesty.org/en/latest/news/2020/06/bahrain-kuwait-norway-contact-tracing-apps-danger-for-privacy/, June 2020. (Accessed on 06/23/2020).
17. Qatari contact-tracing app 'put 1m people's sensitive data at risk'. https://www.theguardian.com/world/2020/may/27/qatar-contact-tracing-app-1m-people-sensitive-data-at-risk-coronavirus-covid-19, 2020. (Accessed on 06/23/2020).
18. Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Alexander Gruler, Jonas Höchst, Joshua Kühlberg, Mira Mezini, Markus Miettinen, Anel Muhamedagic, Thien Duc Nguyen, et al. Mind the gap: Security & privacy risks of contact tracing apps. *arXiv preprint arXiv:2006.05914*, 2020.
19. BBC. Coronavirus: Security flaws found in nhs contact-tracing app. https://www.bbc.com/news/technology-52725810, May 2020. (Accessed on 06/23/2020).
20. Guillaume Celosia and Mathieu Cunche. Fingerprinting bluetooth-low-energy devices based on the generic attribute profile. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, pages 24–31, 2019.
21. Dongyao Chen, Kang G Shin, Yurong Jiang, and Kyu-Han Kim. Locating and tracking ble beacons with smartphones. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 263–275, 2017.
22. Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. Contact tracing mobile apps for covid-19: Privacy considerations and related trade-offs. *arXiv preprint arXiv:2003.11511*, 2020.
23. Aveek K Das, Parth H Pathak, Chen-Nee Chuah, and Prasant Mohapatra. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, pages 99–104, 2016.
24. Ken TD Eames and Matt J Keeling. Contact tracing and disease control. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1533):2565–2571, 2003.
25. Kang Eun Jeon, James She, Perm Soonsawad, and Pai Chet Ng. Ble beacons for internet of things applications: Survey, challenges, and opportunities. *IEEE Internet of Things Journal*, 5(2):811–828, 2018.
26. Douglas Leith and Stephen Farrell. Coronavirus contact tracing app privacy: What data is shared by the singapore opentrace app? In *International Conference on Security and Privacy in Communication Networks*, 2020.
27. Douglas Leith and Stephen Farrell. Gaen due diligence: Verifying the google/apple covid exposure notification api, 2020. SCSS Tech Report.
28. Jinfeng Li and Xinyi Guo. Covid-19 contact-tracing apps: A survey on the global deployment and challenges. *arXiv preprint arXiv:2005.03599*, 2020.
29. Xin-Yu Lin, Te-Wei Ho, Cheng-Chung Fang, Zui-Shen Yen, Bey-Jing Yang, and Feipei Lai. A mobile indoor positioning system based on ibeacon technology. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4970–4973. IEEE, 2015.
30. Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in mobile apps. In *NDSS*, 2018.
31. Leonie Reichert, Samuel Brack, and Björn Scheuermann. Privacy-preserving contact tracing of covid-19 patients. *IACR Cryptol. ePrint Arch.*, 2020:375, 2020.

32. Mike Ryan. Bluetooth: With low energy comes low security. In *Presented as part of the 7th {USENIX} Workshop on Offensive Technologies*, 2013.
33. Lucy Simko, Ryan Calo, Franziska Roesner, and Tadayoshi Kohno. Covid-19 contact tracing and privacy: Studying opinion and preferences. *arXiv preprint arXiv:2005.06056*, 2020.
34. Ruoxi Sun, Wei Wang, Minhui Xue, Gareth Tyson, Seyit Camtepe, and Damith Ranasinghe. Vetting security and privacy of global covid-19 contact tracing applications, 2020.
35. Qiang Tang. Privacy-preserving contact tracing: current solutions and open questions. *arXiv preprint arXiv:2004.06818*, 2020.
36. Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. Decentralized privacy-preserving proximity tracing. `https://github.com/DP3T/documents`. (Accessed on 06/23/2020).
37. Michael Veale. Analysis of the nhsx contact tracing app 'isle of wight'data protection impact assessment. 2020.
38. Mark Weiser. Program slicing. *IEEE Transactions on software engineering*, (4):352–357, 1984.
39. Haohuang Wen, Qi Alfred Chen, and Zhiqiang Lin. Plug-n-pwned: Comprehensive vulnerability analysis of obd-ii dongles as a new over-the-air attack surface in automotive iot. In *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA, August 2020. USENIX Association.
40. Haohuang Wen, Qingchuan Zhao, Qi Alfred Chen, and Zhiqiang Lin. Automated cross-platform reverse engineering of can bus commands from mobile apps. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS'20)*, San Diego, CA, February 2020.
41. Zhemin Yang and Min Yang. Leakminer: Detect information leakage on android with static taint analysis. In *2012 Third World Congress on Software Engineering*, pages 101–104. IEEE, 2012.
42. Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054, 2013.
43. Qingchuan Zhao, Haohuang Wen, Zhiqiang Lin, Dong Xuan, and Ness Shroff. On the accuracy of measured proximity of bluetooth-based contact tracing apps. In *International Conference on Security and Privacy in Communication Networks*, 2020.
44. Qingchuan Zhao, Chaoshun Zuo, Giancarlo Pellegrino, and Zhiqiang Lin. Geo-locating drivers: A study of sensitive data leakage in ride-hailing services. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS'19)*, San Diego, CA, February 2019.
45. Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1469–1483, 2019.