Vetting Security and Privacy of Global COVID-19 Contact Tracing Applications

Ruoxi Sun
The University of Adelaide
Australia

Wei Wang
The University of Adelaide
Australia

Minhui Xue
The University of Adelaide
Australia

Gareth Tyson
Queen Mary University of London
United Kingdom

Seyit Camtepe
CSIRO-Data61
Australia

Damith Ranasinghe
The University of Adelaide
Australia

ABSTRACT

The rapid spread of COVID-19 has made traditional manual contact tracing to identify potential persons in close physical proximity to an known infected person challenging. Hence, a number of public health authorities have experimented with automated contact tracing apps. While the global deployment of contact tracing apps aims to protect the health of citizens, these apps have raised security and privacy concerns. In this paper, we assess the security and privacy of 35 exemplar contact tracing apps using three methodologies: (*i*) evaluate the design paradigms and the privacy protections provided; (*ii*) static analysis to discover potential vulnerabilities and data flows to identify potential leaks of private data; and (*iii*) evaluate the robustness of privacy protection approaches. Based on the results, we propose a venue-access-based contact tracing solution, *VenueTrace*, which preserves user privacy while enabling proximity contact tracing. We hope that our systematic assessment results and concrete recommendations can contribute to the development and deployment of applications against COVID-19 and help governments and application development industry build secure and privacy-preserving contract tracing applications; a vital tool against fighting the pandemic.

1 INTRODUCTION

COVID-19 is now a global pandemic affecting over 200 countries, after its first recorded outbreak in China in December 2019. To counter its spread, numerous measures have been undertaken by public health authorities, *e.g.* quarantining of people, lock-downs, curfews, physical distancing, and mandatory use of face masks.

Identifying those who have been in close contact with infected individuals, followed by self-isolation (so called *contact tracing*) has proven particularly effective [43]. Consequently, contact tracing has emerged as a key tool to mitigate the spread. However, manual contract tracing, using an army of "detectives" is not trivial and has proven challenging for many countries, *e.g.* UK and Italy. Notably, it is difficult due to the *rapid* and *exponential* growth patterns of the virus and the increased demands on qualified human resources. After only 4 months of the pandemic, the number of daily new case has increased nearly 60 fold.[1] Thus, in many countries it has become extremely difficult to perform manual contact tracing [25, 30, 36].

Government authorities around the world, together with industry, have sought to address the challenge by developing contact tracing applications and services. A plethora of apps and services are currently deployed around the globe. These include the Health Code in China [44], the public COVID-19 website in South Korea [20], and the mobile contact tracing apps released in Singapore [35], Israel [42], and Australia [28, 29]. Contact tracing apps operate by recording prolonged and close proximity interactions between individuals by using proximity sensing methods, *e.g.* Bluetooth. The data gathered allows notifications to be generated to inform persons of a potential exposure to the virus.

Proponents argue that the low cost and scalable nature of contract tracing apps make them an attractive option for health authorities. Despite this, contact tracing apps are *not* universally popular, with a number of prominent critics. They have proven particularly controversial due to potential violations of privacy [37], and security consequences from the mass-scale installation of (rapidly developed) apps across entire populations. Despite attempts to alleviate these concerns by both governments and industry, it is well known that the anonymization of individual information is a challenging problem [27]. This study, to the best of our knowledge, performs the first security and privacy vetting of contact tracing apps. We describe our key contributions below:

- We assess the security and privacy of 35 worldwide Android contact tracing applications, listed in Table 4. We discover about 70% of the apps pose potential security risks due to: (*i*) employing cryptographic algorithms that are insecure or not part of best practice; and (*ii*) storing sensitive information in clear text that could be potentially read by attackers. Over 60% of apps pose vulnerabilities through Manifest weaknesses, *e.g.* allowing permissions for backup (hence, the copying of potentially unencrypted application data). Further, we identify that approximately 75% of the apps contain at least one tracker, potentially causing serious privacy leakage, *i.e.* data leaks that lead to exposing private information, to third parties.

- We analyze user privacy exposure and privacy protections provided by 10 solutions—covering 3 different frameworks

---

[1]1,354 confirmed per day in the first 30 days from 11 January to 10 February 2020 and 80,487 confirmed per day in the recent 30 days from 18 April to 17 May 2020—using data released by the World Health Organization (WHO).

(*PACT* [24], *Covid Watch* [49], and *PEPP-PT* [51]), the Coronavirus Disease-19 website [20], and 6 applications — from 7 countries around the world. We establish a threat model and analyze the vulnerabilities of the apps to multiple privacy attacks. The results demonstrate that there is no solution that is able to protect users' privacy against all of attacks investigated. Generally, Bluetooth-based decentralized solutions that avoid direct location tracking outperform centralized systems.

- We synthesize the findings from our extensive COVID-19 contact tracing app vetting exercises to: (*i*) provide best practice security and privacy guidance to governments and app industry; and (*ii*) recommend a novel decentralized venue-accessing-based contact tracing approach, termed *Venue-Trace*, to overcome potential privacy issues highlighted in the state-of-the-practice solutions. Our *VenueTrace* proposal has the capability to significantly increase the privacy protections for citizens whilst being securely implemented.

We have disclosed our findings and detailed security and privacy risk reports to the related stakeholders. We believe our study can provide useful insights for governments, developers and researchers in the software industry to develop secure and privacy-preserving contact tracing apps. We hope the results and the proposed contact tracing approach will contribute to increasing the trustworthiness of solutions to contain infectious diseases now and in the future.

2 CONTACT TRACING APPLICATIONS

A range of contract tracing applications (or "apps") are used worldwide. Given the large number of contact tracing apps and proposed solution frameworks, we survey a representative sample to more broadly study the architectures employed, design paradigms and the privacy exposure of the user groups we identified in Section 2.1.

2.1 Users Groups and Privacy Exposure

We define model user groups and privacy exposure levels to aid our our investigations into app architectures (in Section 2.2) and privacy vetting (in Section 4). We envisage three groups of contact tracing app users, based on their health status. We describe the three users groups below and analyze their privacy exposure in Section 2.2.

- **Generic user.** A typical user of the contact tracing system, who is healthy or has not been diagnosed yet.
- **At-risk user.** Alice, who has recently been in contact with an infected user, Bob. Ideally, Alice will receive an at-risk alarm from her application.
- **Diagnosed user.** As a diagnosed patient, Bob will be asked to reveal his private information as well as the information of at-risk users to the health authorities, *e.g.* the diagnosis of his infection, his movement history, the persons he has been in contact with.

We define user group exposures in different apps using five levels:

- Level I: "No data shared", the most secure level in which there is no user data shared.

- Level II: "Tokens are shared with proximity users", a medium exposure level with only tokens containing no Personal Identifiable Information (PII) exchanged between users.
- Level III: "Tokens are shared with the server", a medium exposure level with tokens exposed to the server.
- Level IV: "PII is shared with a server", a high risk exposure level in which the users' PII is shared with the server.
- Level V: "PII is published to public", the highest risk exposure level.

2.2 Analysis of Design Paradigms and User Privacy Exposure

We select 10 well known contact tracing solutions—including both current apps and proposed frameworks. Table 1 presents an overview of the 10 selected solutions. Four of the selected solutions are developments of apps used from the early stages of the pandemic (supported by governments such as China, South Korea, Singapore, and Israel). The one service and a proposed framework from Europe were selected because they are the first solutions that enable anonymous identifier exchange. We have also selected three solutions that are about to or have already been deployed from North America and one app deployment from Oceania. As summarized in Table 2, all 10 solutions have user privacy exposure to some extent. We next discuss them in the context of the two broad categories of: (*i*) centralized architectures; and (*ii*) decentralized/distributed architectures.

Centralized solutions. Many solutions utilize a centralized system in which the central server is responsible for: (*i*) information collection; and (*ii*) health status evaluation for users. In some East Asian countries, *e.g.* China and South Korea, where the outbreaks first occurred, contact tracing systems were quickly developed and released. The systems helped health authorities to successfully control the spread of COVID-19, but a huge amount of Personal Identifiable Information (PII) was collected.

In South Korea, the *Coronavirus Disease-19 website* [20] (#7 in Table 1) is supported by the Ministry of Health and Welfare. Although Alice's privacy is protected as no data is required from her, the system publishes Bob's information to the public (marked as Level V in Table 2). The information exposed includes gender, nationality, age, diagnosis date, hospital, and movement history (removed in the latest version). This directly puts Bob at risk of being re-identified, raising serious privacy concerns. For example, as reported by The Washington Post [15], in Cheonan, a city in South Korea, a text alert to residents showed that an infected person visited "Imperial Foot Massage at 13:46 on 24 February".

In China, *QR-code contact tracing* apps were developed by the two Tech companies, Alibaba and Tencent [44] (#2). The apps use a colour code to present the health condition of an individual–green implies people can travel freely, while yellow or red indicates they must report to the authorities. Users need to provide their name, national ID, and phone number to register and use the app to enter public places, *e.g.* the metro stations, supermarkets, and airports. The apps are mandatory and jointly developed by government departments and supported by data from health and transport

Table 1: Representative state-of-the-practice solutions from seven countries and five continents.

ID	Country	Name	Developer	Private/State	Technique	Architecture
1	Australia	COVIDSafe	Australian Department of Health	State	Bluetooth	Centralized
2	China	Health Code	Alibaba	Private	QR code	Centralized
3	Europe	PEPP-PT*	International consortium	Private	Bluetooth	Centralized
4	Europe	DP3T	International consortium	Private	Bluetooth	Decentralized
5	Israel	HaMagen	Ministry of Health	State	Location	Decentralized
6	Singapore	TraceTogether	Government Technology Agency	State	Bluetooth	Centralized
7	South Korea	Coronavirus Disease-19	Ministry of Health and Welfare	State	Location	Centralized
8	USA	Covid Watch*	Standford University	Private	Bluetooth	Decentralized
9	USA	Private Kit	Massachusetts Institute of Technology	Private	GPS+Bluetooth	Decentralized
10	USA	PACT*	University of Washington	Private	Bluetooth	Decentralized

*Note: frameworks with no application implemented

Table 2: User exposure.

Solutions	Generic	At-risk	Diagnosed	Architecture
COVIDSafe	Level II	Level IV	Level IV	Centralized
Health Code	Level IV	Level IV	Level IV	Centralized
PEPP-PT	Level II	Level III	Level III	Centralized
DP3T	Level II	Level II	Level III	Decentralized
HaMagen	Level I	Level I	Level V	Centralized
TraceTogether	Level II	Level IV	Level IV	Centralized
Coronavirus Disease-19	Level I	Level I	Level V	Centralized
Covid Watch	Level II	Level II	Level III	Decentralized
PACT	Level II	Level II	Level III	Decentralized
Private Kit	Level II	Level II	Level III	Decentralized

Level I ▪: No data shared,
Level II ▪▪: Token shared with proximity users, Level III ▪▪▪: Token shared with the server,
Level IV ▪▪▪▪: PII shared with the server, Level V ▪▪▪▪▪: PII is published to public.

authorities. In this solution, for all types of users, their privacy information will be shared with the central server. Thus, we evaluate the user exposure of Health Code as Level IV in Table 2.

TraceTogether [35] from Singapore is the first solution that uses Bluetooth technology. Bluetooth-based solutions rely on proximity tracing via Bluetooth broadcasts from apps. As these occur exclusively between devices in proximity, these methods provide more scope for privacy-preserving computations compared to those that use GPS locations (*Coronavirus Disease-19* and *Hamagen* [42]). In TraceTogether, proximity between two users is measured through the Bluetooth broadcast signals and encrypted user information is stored on mobile devices. Once diagnosed, the user will be asked to upload their local on-device records to the Ministry of Health with the authority to decrypt the data and obtain the mobile numbers of the user's close contacts within a period of time (*e.g.* 21 days) that covers the incubation period of the virus. Such a centralized BLE-based solution preserves more personal privacy as the data exchanged between users is not related to absolute location information. *COVIDSafe* [29] from Australia also utilizes a similar technique. However, considering that the PII, *e.g.* phone numbers, is collected by the government, the at-risk and diagnosed users' PII is exposed to the central server (Level IV in Table 2), while the exposure of generic users still remains in Level II as the tokens are only shared between users.

The solutions that expose PII may not work well in countries with different societal norms. Thus, many western countries developed solutions with no PII related information exchange, *e.g.* PEPP-PT [51]. In PEPP-PT, an ephemeral user ID is implemented based on a seed randomly generated by a user device. Users will exchange ephemeral IDs, instead of encrypted PII messages, to record a proximity contact event, thus reducing the privacy exposure of diagnosed users to Level III where tokens are shared with servers.

Decentralized solutions. The second type of solution is decentralized, where: (*i*) the back-end server is only responsible for collecting data from diagnosed patients; and (*ii*) the health evaluation is conducted on users' devices, locally. This design prevents the central server from knowing the infected person, and their physical proximity contacts.

Some decentralized systems rely on location information. For example, *Hamagen* (#5), an app provided by the Israeli Ministry of Health, obtains but does not share location data from the user's phone and compares it with the information stored in a central server regarding the location histories of confirmed cases. As no data is shared before diagnosis, the exposure level of at-risk users are evaluated as Level I in Table 2. However, the diagnosed user will be notified and given the option of reporting their exposure to the Health Ministry by filling out a form; subsequently, their location trail is released to public.

Other Bluetooth solutions, *e.g. DP3T* [34] (#4), *Covid Watch* [49] (#8), and *PACT* [24] (#10), implement decentralized designs to allow users to download diagnosed anonymous identifiers from the back-end server and compare with local records obtain their risk of exposure to the virus. The design paradigm reduces the exposure level of at-risk users to Level II and the level of diagnosed users to Level III in Table 2 as no PII is shared by users.

Another application, *Private Kit* [11] (#9), a decentralized solution developed by Raskar *et al.* [46], enables individuals to log their own location information. Particularly, Private Kit also allows Bluetooth broadcasts between users to enable direct notification between users. As the sharing of diagnosed users' location trail and the broadcasts between users is privacy protected, we determine Private Kit's user exposure levels to be the same as other Bluetooth-based decentralized solutions. We will discuss in Section 3.3 how the increasing of functionalities affect the privacy of users.

3 SECURITY VETTING

In this section, we consider the list of 35 contact tracing apps curated and summarised in Table 4. We downloaded the apps from Google Play Store and evaluated their security performance against the four vetting categories: (*i*) manifest weaknesses; (*ii*) data leaks (with a focus on those that violate user privacy); (*iii*) Malware; and
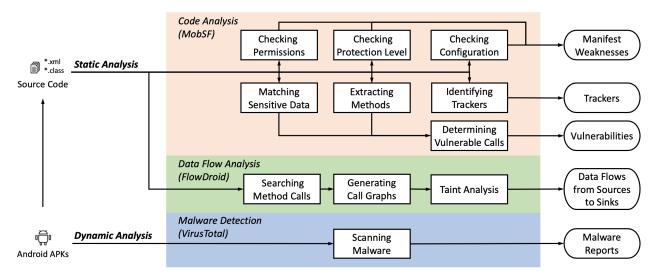
Figure 1: Overview of our security vetting methods. Importantly, we also augment the analysis with manual inspections.

Table 3: Security vetting categories.

Vetting Category*	Security Issues
Manifest Weaknesses	Insecure flag settings, *e.g.* app data backup allowed Non-standard launch mode Clear text traffic
Privacy Leaks	Trackers Potential Leakage Paths from Sources to Sinks
Malware	Viruses, worms, Trojans and other kinds of malicious content
Vulnerabilities	Sensitive data logged SQL injection IP address disclosure Uses hard-coded encryption key Uses improper encryption Uses insecure SecureRandom Uses insecure hash function Remote WebView debugging enabled

*Note: security issues above are extracted and summarized from the tools FLOWDROID (Privacy Leaks), VIRUSTOTAL (Malware), and MOBSF (Manifest Weaknesses, Privacy Leaks, & Vulnerabilities).

(*iv*) general vulnerabilities. We detail the list of issues considered in Table 3.

3.1 Methodology

An overview of our security vetting methodology is shown in Figure 1 and we describe the method used for selecting the apps for our investigation in Section 3.1.1. We perform: (*i*) static analysis, including code analysis and data flow analysis;(*ii*) dynamic analysis to detect malware.

3.1.1 Apps Selection. In order to curate a list of contact tracing apps, we first searched the keywords, *e.g.* "contact tracing", "Covid", "tracing coronavirus", in Google Play Store. We also started with known official apps from countries, *e.g.* the COVIDSafe recommended by Australian government. After a contact tracing app is found, we assess its functionality by reading the app descriptions and select those with in excess of 10,000 downloads. Subsequently, we include the app into the set and look for new apps through the recommendation links in the app store, until there are no contact

Table 4: Contact tracing apps considered in our analysis.

Applications*	Country	Downloads	Version
Coronavirus AlgÃŤrie	Algeria	100K	1.0.3
Stopp Corona	Austra	100K	1.1.4.11
CoronaReport	Austra	10K	2.9.5
Coronavirus Australia	Australia	500K	1.0.2
COVIDSafe	Australia	1M	1.0.11
BeAware Bahrain	Bahrain	100K	0.1.4
Coronavirus Bolivia	Bolivia	50K	1.2.7
Coronavirus - SUS	Brasil	1M	2.0.5
COVID-19!	Czech	10K	0.9.4
Stop Covid	Georgia	100K	1.0.461
COVA Punjab	India	1M	1.2.2
CG Covid-19 ePass	India	500K	1.0.6
West Bengal Emergency Fund	India	10K	1.3
patientMpower for COVID-19	Irish	10K	1.1.1
Hamagen	Israel	1M	1.1.2
Stop COVID-19 KG	Kyrgyzstan	10K	0.3.137.325
MySejahtera	Malaysia	500K	1.0.8
SOS CORONA	Mali	10K	0.0.6
Nepal COVID-19 Surveillance	Nepal	5K	1.1.1
Hamro Swasthya	Nepal	50K	1.3.2
COVID Radar	Netherlands	50K	1.1.2
National Action Plan	Pakistan	50K	1.1
Corona Map	Saudi Arabia	50K	1.0.0
TraceTogether	Singapore	500K	1.8.0
NICD COVID-19 Case Investigation	South Africa	10K	1.16
STOP COVID19 CAT	Spain	500K	1.0.2
StopTheSpread COVID-19	UK	100K	1.0.0
Coronavirus UY	Uruguay	100K	2.2.3
Private Kit	USA	10K	0.5.19
Contact Tracing	USA	10K	1.3.8
Contact Tracer	USA	10K	2.0.2
COVID-19	Vietnam	100K	1.0
NCOVI	Vietnam	1M	1.5.3
Vietnam Health Declaration	Vietnam	100K	1.0.12
Bluezone	Vietnam	100K	1.0.1

*Note: apps were collected on 24 April 2020 and 1 May 2020 from Google Play Store.

tracing apps found. We repeated this procedure one week later and finalised the list of 35 contact tracing apps summarised in Table 4 on 1 May 2020.

3.1.2 Static Analysis. We perform static analysis on the Android Package (APK) binary files. We first de-compile the APK of each app
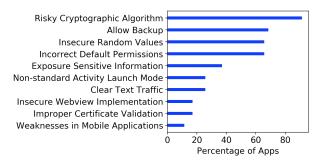
Figure 2: Code analysis results.

to its corresponding *class* and *xml* files. Then, we utilize the Mobile Security Framework (MobSF) [8] to perform code analysis and FlowDroid for data flow analysis. Notably, we augment our static analysis with manual inspections to further increase the robustness of the vetting process. We detail our approach in Appendix A.

3.1.3 Dynamic Analysis. We rely on malware scanners to flag malicious artifacts in contact tracing apps. Concretely, we send the APKs to VirusTotal [18], a free online service that integrates over 70 antivirus scanners, which has been widely adopted by the research community [33, 41]. As shown in Table 3, the results of malware detection will identify the detected viruses, worms, Trojans, and other malicious content embedded in the apps.

3.2 Security Vetting Results

We next inspect the presence of security vulnerabilities among the 35 considered apps. Figure 2 shows the number of contact tracing apps that have security weakness found in our Code Analysis.

Code analysis. Figure 2 shows that the most prominent vulnerabilities extracted from the manifest weaknesses. We observed that 68% of apps do not set the flag `allowBackup` to `False`. Consequently, users with enabled USB debugging can copy application data from the device. Other weaknesses detected are related to "Clear Text Traffic" such as plaintext HTTP, FTP stacks, DownloadManager, and MediaPlayer, which may enable a network attacker to implement man-in-the-middle (MITM) [7] attacks during network transmission.

Notably, during our manual review of the vetting results from MobSF. We found three false positives results, *i.e.* Clear Text Storage, Incorrect Default Permissions, and SQL Injection. For example, in the application COVIDSafe, broadcast and channel identifiers, encryption algorithm names, and placeholders, which are used to receive or query specific values but do not contain sensitive information, are stored as constant values. However, MobSF regards all constant string values as potential clear text storage. Some applications, *e.g.* Coronavirus UY (Uruguay), create template files while decompressing and loading multiple dex files in order to avoid the 64K reference limit [16]. Other applications, *e.g.* CG Covid-19 ePass (India), are able to scan other's barcode and save it into temporary files in order to read the content. However, these behaviours are mis-regarded as temporary file leakage by MobSF. All these false-positives were removed through manual inspections from our further analysis.

Table 5: Trackers identified in contact tracing apps.

Trackers	# of Apps	Percentage
Google Firebase	25	71.4%
Google CrashLytics	6	17.1%
Other Google trackers	4	11.4%
Facebook trackers	3	8.6%
Other trackers	9	25.7%

Figure 2 shows that the most frequent weakness detected by static analysis is the "Risky Cryptography Algorithm". Over 91% of apps use at least one of the algorithms, *e.g.* MD5 and SHA-1. For instance, in the app MySejahtera (Malaysia), the parameters in WebSocket requests are combined and encrypted with MD5 which will be compared with the content from requests in the class `Draft_76` in order to verify the validity of connections. Although this has been listed in the top 10 OWASP [9] mobile risks 2016, the results show that it is still a common security issue. Another frequent weakness is "Clear Text Storage" (files may contain hard-coded sensitive information like usernames, passwords, keys *etc.*). In class `DataBaseSQL` of COVID-19 (Vietnam) app, the password of SQLite database is stored in the source code without encryption; CG Covid-19 ePass (India) also hard-coded its encryption key in class `Security`.

In total, 20 trackers have also been identified, including `Google Firebase Analytics`, `Google CrashLytics`, and `Facebook Analytics`. Approximately 75% of the apps contain at least one tracker. As shown in Table 5, the most frequent tracker is `Google Firebase Analytics` which is identified in more than 70% of the apps. Notably, the research study in [38] argues that TraceTogether using Google's Firebase service to store user information may leak user's privacy to third parties, such as Google. In the most extreme case, a contact tracing app, the Contact Tracing (USA), contains 8 trackers.

Data flow analysis. Figure 3 presents the flow of data between sources and sinks. This is counted by the number of source-to-sink paths found in each apps. The top sources of sensitive data are methods calling from `Location` and `database.Cursor`. These may obtain sensitive information from a geographic location sensor or from a database query. Most of the sensitive data will be transferred to sinks, such as `Bundle`, `Service`, and `Log`, which may leak sensitive information out of apps. As discussed previously, sending sensitive information to the `Bundle` object may reveal sensitive data to other activities. Besides, developers usually utilize `Log` to print debugging information into `Logcat` [6] panel, however human errors from developers can lead to mistakenly printing sensitive data. Notably, we discovered that some apps transmit location information through SMS messages. Considering Hamagen as an example, location information is detected and obtained by a source method `initialize(Context,Location,e)` and then flows to a sink method where `Handler.sendMessage(Message)` is called. This a potential vulnerability as malware could easily intercept the outbox of Android SMS service [21]

We also manually vetted the FlowDroid results for false positives. In total, 60 out of 371 paths (16.17%) were false positives (results presented in Figure 3 are excluding these false positives).
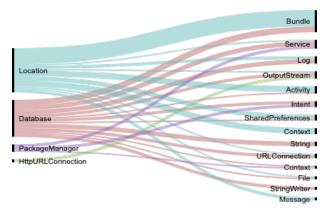
Figure 3: Overview of COVID-19 app data flow analysis.

There are mainly two categories of false positives. The first one is related to "Log" sinks where FLOWDROID marks all log methods as sinks, while some of them are not actually sensitive. For instance, in TraceTogether, error messages such as `SQLiteException` from the stack trace occurred while data querying will be logged by `Log.e` method. This matches the keywords and is a false-positive identified as a sink. Another example is that in Private Kit: while the status of LOCATIONPROVIDER changes, geo location data are read in the implemented function of LOCATIONLISTENER.ONSTATUSCHANGED. By comparing to decompiled source code, we found only the status names, including `OUT_OF_SERVICE`, `TEMPORARILY_UNAVAILABLE`, and `AVAILABLE` are logged by `Log.v` or `Log.d` instead of geo-location data. Most of the false positives we find fall in this category. Another type of false positive results from Preference Leakage detection. For example, the app STOP COVID19 CAT, stores country code (*e.g.* UK and AU) in the `SharedPreference`, fetched by invoking `Locale.getCountry` method. However, this is not confidential and does not leak privacy.

Malware detection. We discovered only one application containing malware, `Stop COVID-19 KG`.[2] Two risks are identified: a variant Of `Android/DataCollector.Utilcode.A` and an Adware (0053e0591). Consider the limited spreads of this app (about 10,000 downloads), we conclude that the vast majority of contact tracing apps are free of malware.

3.3 Case Studies

From our curated list of 35 apps, we select four typical app to further highlight key lessons we can learn with respect to security and privacy risks. The case studies are based on TraceTogether, DP3T, Private Kit, and COVIDSafe.

TraceTogether. This mobile application released by the Singapore Government allows for digital contact tracing using the custom BlueTrace [35] protocol. According to the static analysis results, root detection [32] has been implemented, which potentially prevents SQL injection and data breaches, thereby reducing the the risk to a certain extent.

In the decrypted class `o/C3271ax.java`, root detection logic is implemented by detecting the existence of specific root files. Files

[2]https://play.google.com/store/apps/details?id=kg.cdt.stopcovid19

including `/system/app/Superuser.apk` and `/system/xbin/su`, only exist in a root device; by assessing their integrity, the application can detect whether a device is rooted and subsequently block users from either login or opening the applications.

> **Security guidance 1:** To protect users' devices against being controlled by malicious software, root detection [1] and confidential data protection [14] should be enabled at application startup.

However, TraceTogether also includes a third-party customer feedback library, ZENDESK SDK, in which the remote WebView debugging is enabled. This potentially allows attackers to dump the content in the WebView [23]. When a user inputs confidential data, including passwords and identity information, in a debug-enabled WebView, attackers may be able to inspect all elements in the web page by using remote debug tools [13]. Fortunately, according to the static analysis, the only WebView with debugging mode enabled is to display articles; therefore, does not contain confidential data.

> **Security guidance 2:** Never leave WebView with debugging mode enabled in the App release.

DP3T. According to the static analysis, DP3T's database is not encrypted, and data is saved in plain text. In contrast to TraceTogether, the app does not implement any root detection capabilities. This means that a malicious app could possibly access the database directly and manipulate the database containing COVID-19 contact records. Potentially, an adversary could spread false-positive.

> **Security guidance 3:** To protect the database from being dumped and prevent data breaches, database encryption must be implemented [17].

In addition, as the database records timestamps and contact IDs, the leakage of the database from a root device infected by a mobile system virus can be exploited to mount linkage attacks by adversaries [12]. If enough data in a region were collected by attackers, contact IDs and timestamps in the database can be used to analyze movements by comparing data and device owners may be identified through a linkage attack [56].

Private Kit.

Similar to DP3T, Private Kit does not encrypt the database and contains plaintext data. Besides, the app creates temporary JSON files to store user's location data. Without any encryption and root detection, the temporary JSON files can be dumped from root devices; thus increasing the risk of privacy leakage.

> **Security guidance 4:** To prevent potential data breaches, tracing records and confidential data must not be stored in temporary files in plaintext.

COVIDSafe.

COVIDSafe is a digital contact tracing app announced by the Australian Government [3]. According to our experiments, COVIDSafe stores all tracing histories, including contacted device IDs and timestamps, into SQLite database with plain text. Since the application does not implement a root detection logic, tracing histories

may be leaked from root devices and potential Linkage Attacks can be implemented [56].

Besides the issue mentioned above, there are at least two core issues being discussed [5, 10].The first issue is that the COVIDSafe app caches the Bluetooth characteristic value that will be read by a remote device using the MAC address of the remote device. Due to a logic error, this cache is only cleared when a successful transaction takes place [2]. The second issue is much simpler. The advertising payload is generated once at startup and used for the lifetime of the app. Due to the requirements by the iPhone app, there are additional bytes included in this payload added to the manufacturer field. However, these bytes are generated once at startup, and then the same data is used continuously for the lifetime of the app. This means that the advertising payload will consistently identify the same device [2].

4 PRIVACY RISK ASSESSMENT

In this section, we describe the privacy analysis we conducted on the 10 selected contact tracing solutions in Table 4 to assess their protection against potential privacy breaches under our threat model.

4.1 Threat Model

We consider four attackers in our threat model in addition to the user groups defined in Section 2.1:

Application users. Those who install contact tracing applications on their mobile phones will receive information about COVID-19, *e.g.* an at-risk alarm. A regular user may reveal their private information, *e.g.* name, gender, phone number, national ID, home address, and location history, to the contact tracing systems, as well as discover other users' private information from the system—pubic information or broadcasts from other users.

Health authorities. The actors are responsible for diagnosing infections and collecting health information from Application users. May learn or deduce private information about at-risk users. Health authorities will also help the diagnosed users record or upload information to the contact tracing system.

Governments. These actors work with technology providers and are often responsible for operating the contact tracing system. They may access the data stored in a central server. In our threat model, we suppose the Government (and even the cloud operator) is "untrusted", that is, they may use the collected data for purposes beyond the pandemic.

Malicious adversaries. These adversaries have access to local app information. They follow the defined algorithms, but wish to learn more than the allowed information. They may have the capability to access the local log of contact tracing applications, but hacking the back-end server or another user's device is out of the scope of their capabilities. They may utilize some devices, such as a Bluetooth broadcaster or receiver, to attack the system or gain extra information. They may also modify the app and impersonate a legitimate user to access the system, which is difficult to prevent unless remote attestation is applied.

4.2 Potential Privacy Attacks

As discussed previously, the privacy of users is hard to preserve in a contact tracing system. To introduce potential privacy risks, we will let *Alice* be an at-risk user, and let *Bob* be a diagnosed user who has been in contact with Alice. *Mallory* will be a malicious attacker, and *Grace* will be the government server (or other authority). Here we discuss four privacy attacks; the privacy vetting results are summarized in Table 6.

Linkage attacks by servers. In centralized systems, the major privacy concern is metadata leakage by the server. For example, in Coronavirus Disease-19 website, TraceTogether and COVIDSafe, a central server is used to collect PII information and to evaluate at-risk individuals. Consequently, Grace will be able to collect a large amount of PII, such as names, phone numbers, contact lists, post code, home addresses, location trails. Therefore Grace is able to deduce the social connections of Alice. Even for PEPP-PT, a centralized Bluetooth system with solutions to avoid PII collection, the re-identifiable risks still exist. For example, from the server side, Grace is able to link ephemeral IDs to the corresponding permanent app identifier and thus trace Alice based on IDs observed in the past, as well as tracing future movements. Thus, no centralized solutions in Table 1 can prevent linkage attacks by the server.

In contrast, for decentralized solutions, Alice's privacy is protected as there is no information sent to a central server and her health status is evaluated on her own device. Thus, decentralized systems are able to protect users' privacy against linkage attacks by the server, except Hamagen, where the server learns Bob's location trail.

> **Privacy guidance 1:** To protect users' privacy against linkage attacks by a server, a contact tracing solution should:
> (1) Avoid sharing PII information with central points
> (2) Implement a decentralized design

Linkage attacks by users. Linkage attacks, performed by Mallory, try to re-identify Alice or Bob. As discussed previously, in contact to tracing systems that directly publish users' PII, *e.g.* Coronavirus Disease-19, Hamagen, and Health Code, Bob is at the risk of privacy leakage. For example, in Coronavirus Disease-19, Mallory could be able to re-identify Bob as he will know Bob's gender, age, and location history from public information.

The other 7 systems listed in Table 1 further rely on information exchange between users. In number of the apps, *e.g.* DP3T which implement an ephemeral ID design, Mallory is still able to identify Bob using more advanced attacks. For example, if Mallory places a Bluetooth receiver near Bob's home or working place and ensures that the device will only receive Bluetooth broadcasts from Bob. Once Bob is diagnosed, Mallory will receive an at-risk alarm and immediately acknowledge that the infected patient is Bob. In addition, Mallory can enable tracing on multiple users at the same time by logging the timestamp and the received ephemeral ID when in contact with Bob.

Once Bob is diagnosed, Mallory can trace back the source of recording and re-identify Bob and potentially infected users. Even worse, if Mallory distributes multiple broadcast receivers in a large area with some layout, *e.g.* honeycomb, they could even trace the movement of Bob by tracing the records on each device.

Thus, none of the 10 typical solutions can fully protect users' privacy against linkage attacks by Mallory.

> **Privacy guidance 2:** To protect users' privacy against linkage attacks by an adversary, a solution should:
> (1) Avoid data sharing between users
> (2) Ensure privacy protections exist for any published data.

False positive claims. In some systems, such as Coronavirus Australia, Bob can register as infected and upload data through the contact tracing app to the server, which enables Alice to receive an at-risk alarm. However, if Mallory exploits such a mechanism and registers as a (fake) infected user, Alice will receive a false-positive at-risk alarm, which may cause social panic or negatively impact evidence-driven public health policies. Most solutions mitigate this issue by implementing an authorization process, *i.e.* Bob is only permitted to upload data after receiving a one-time-use permission code generated by the server. Without the permission code, Mallory is not allowed to claim they are infected and Alice will always receive a true at-risk alarm. Only two solutions, *i.e.* DP3T and PACT [24], have no authorization process implemented.

> **Privacy guidance 3:** To protect a system against false-positive-claim attacks, a solution should establish an authorisation process.

Relay attacks.[3] To apply such an attack, Mallory could collect existing broadcast messages exchanged between users, then replay it at another time or forward it through proxy devices to a remote location and replay the messages. Due to the lack of message validation in solutions that utilize information broadcasts, a user will not be able to determine whether a received broadcast is from a valid source or from a malicious device. Any received broadcast will be recorded as a contact event, even though no actual contact exists.

For example, suppose Mallory records the broadcasts from Bob and then replays it hours later, or transmits it to a remote location and replays the messages to Alice. Alice's device, although not actually being in contact with Bob, will receive and record the replayed broadcast in its local log. Once Bob is diagnosed, Alice will receive an at-risk alarm even though she has never been in contact with Bob. Such an attack will falsely enlarge the contact range of Bob and create a large amount of false-positive alarms, which may cause panic among citizens. Solutions that do not utilize information broadcasts, such as Coronavirus Disease-19 and Hamagen, can avoid relay attacks.

One promising solution is to use an ambient physical sensing approach, *e.g.* ambient audio. This has been shown to secure proximity detection [31, 50] by comparing the ambient information embedded in the broadcast messages with the local ambient. It allows a receiver to validate whether the source is nearby as the range of Bluetooth broadcast is generally within 50 m.

> **Privacy guidance 4:** To protect a system against relay attacks, a solution should:

---

[3]The combination of man-in-the-middle and replay attacks are henceforth referred to as a relay attack.

Table 6: Privacy protections against the vetting attacks.

Solutions	Linkage-Server	Linkage-User	False-Claim	Relay
CovidSafe	●	●	○	●
Health Code	●	●	○	○
PEPP-PT	●	●	○	●
DP3T	○	●	●	●
HaMagen	●	●	○	○
TraceTogether	●	●	○	●
Coronavirus Disease-19	●	●	○	○
Covid Watch	○	●	○	●
Private Kit	○	●	○	●
PACT	○	●	●	●

○: the privacy is well protected ●: the privacy is at risk

> (1) Either avoid utilizing information broadcast or
> (2) Implement a validation approach

5 OUR RECOMMENDATIONS

As discussed in Section 4, a contact tracing application should preserve the privacy of generic and at-risk users. Although the diagnosed user may reveal their privacy to health authorities, we should not release this data to the public. Furthermore, we argue that a contact tracing solution should focus on tracing anonymous daily routines or occasional contacts, instead of close contacts. Here we propose a venue-accessing-based solution, *VenueTrace*, to overcome privacy risks. We will first describe the framework of this solution and assess its privacy performance as well as its limitations.

5.1 Our Privacy-by-Design: *VenueTrace*

The architecture of *VenueTrace* contact tracing consists of the following modules as described in Figure 4.
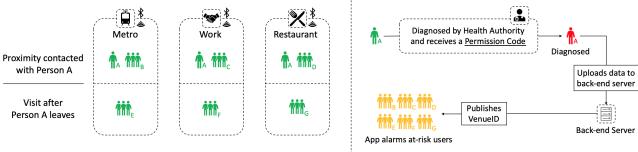
Bluetooth broadcaster in public places. Instead of broadcasting from each user's mobile phone, the *VenueTrace* proposes utilizing Bluetooth broadcast devices installed in public places, *e.g.* restaurants, movie theaters, working places, and public transport hubs and stops. This is in contrast to most existing Bluetooth solutions that rely on human-to-human contact. To facilitate contact tracing, at setup, each broadcast device will register its MAC address to a back-end server and get a unique *VenueID* which will be broadcast through Bluetooth at every *Time Interval T*. Hence, users are not required to broadcast information.

Applications installed in the user's phone. For every VenueID broadcast by a device in its proximity at time t, after receiving the broadcast VenueID, a pair (ID, t) will be created and a timer will be started in the user application. If a user receives the VenueID after T again, the user stores a tuple which satisfies the following:

$$(ID, t_{start}, t_{end}), \quad \text{where} \quad t_{start} - t_{end} \geq T, \qquad (1)$$

where t_{start} is the first timestamp of the received the broadcast VenueID in the local storage for 14 days; t_{end} is the last timestamp of a period over which a user continuously received VenueID. For example, if T is set to 10 minutes and Bob stays in a public place for more than 30 minutes, he will at least receive the broadcast for three times, *e.g.* t_1, t_2, and t_3. The application will record the tuple

Figure 4: Overview of *VenueTrace* framework.

(ID, t_1, t_3) in local log, where $t_3 - t_1 = 20$ minutes, indicating that he stayed in a public place for at least 20 minutes.

Once Bob is diagnosed, with his consent, he will receive a permission code from the health authority, then Bob can upload his log $(ID_{Bob}, t_{sBob}, t_{eBob})$ and the permission code to the back-end server. To ensure the security of data transmission, the data will be encrypted with the *Public Key* of the back-end server. Every twenty-four hours, Alice will download logs in a format of $(ID_{Server}, t_{sServer}, t_{eServer})$ from the back-end server and a record match and an evaluation will be conducted locally. A local record, *i.e.* $(ID_{Alice}, t_{sAlice}, t_{eAlice})$, is considered as matched if the following conditions are true.

$$ID_{Alice} = ID_{Server}$$
$$(t_{sAlice}, t_{eAlice}) \subseteq (t_{sServer}, t_{eServer}), \tag{2}$$

which indicates that Alice has been in a public place during the period $(t_{sServer}, t_{eServer})$ and may have been in an at-risk environment. If there is a match in Alice's local log, Alice's device will generate an at-risk alarm.

Decentralized back-end server. The back-end server supports the activities of (*i*) registering the Bluetooth broadcaster by storing its MAC addresses and VenueIDs; (*ii*) generating and authorizing permission codes to health authorities; (*iii*) publishing the public key and deciphering the data uploaded by diagnosed users with its *private key*; (*iv*) validating the received VenueIDs and permission code; and (*v*) publishing at-risk information to regular users in the format of tuples $(ID_{Server}, t_{sServer}, t_{eServer})$,

$$ID_{Server} = ID_{Bob},$$
$$t_{sServer} = t_{sBob} - T - random,$$
$$t_{eServer} = t_{eBob} + \delta t + random' \tag{3}$$

where the at-risk time interval is extended based on Bob's record by T in the upper limit and δt in the lower limit. A typical δt could be set to 12 hours to ensure that Alice will be informed that she has been in an at-risk venue where she may have touched a virus-contaminated surface or inhaled airborne droplets, even though she has never been in close physical contact with Bob. We could also further extend the visiting duration with random noise to blur the timeline. For example, if Bob visited a public place from 9 am to 10 am, the released infected duration could be from 8:30 am to 11:15 pm,

where $T = 5\ mins, random = 25\ mins, \delta t = 12\ hours, random' = 75\ mins$. Considering the time-related functionality of a public place, this duration could be further capped.

5.2 Defending Against Attacks

A major flaw in many implemented and proposed applications is that they cannot fully guarantee user privacy when using user-provided data. Of the applications discussed in this paper, this issue is particularly prevalent with centralized solutions, *e.g.* TraceTogether, as they may suffer from linkage attacks not only by users but also by the server. However, any application that requires symptom reporting from its user base could potentially be vulnerable, as discussed in Section 4.

Decentralized computation. Compared to centralized systems, our solution has the inherent advantage of decentralized systems, that is, users' privacy is not exposed to the server. The back-end server will only receive the timestamp and VenueID of a public place that is visited by diagnosed users. Supposing a malicious attacker successfully extracts data from the back-end server, he is still not able to link the information to any location or users as there is no location information stored in the server. Thus, the user privacy is protected against the linkage attack by a server.

Coarse-grained location. Furthermore, in contrast to location based solutions (*e.g.* Hamagen) our solution does not utilize GPS information as Bluetooth is more advantageous than GPS signals in high risk indoor environments. However, considering the extension and blurring in timelines, the back-end server will only receive and publish venue IDs with at most coarse-grained location information. In addition, our solution overcomes the limitation of location-based tracing by installing the broadcaster in public venues and transport in contrast to user devices.

No token exposure. Many Bluetooth-based decentralized systems provide a privacy preserving solution by only sharing the temporary tokens between users. However, it is still amenable to linkage attacks by users. Compared to other decentralized solutions, our solution further preserves user privacy as no information is exchanged between users. Consequently, our system is immune to linkage attacks by users. In the worst case, the attackers may physically visit public places and record the venue IDs, then they may link venue IDs to locations. After an at-risk alarm, the attacker may

perceive where a diagnosed user appeared. However, the attackers still does not have enough information to re-identify the infected individuals, unless they log all persons having appeared in multiple public places for a long period and is able to infer the persons matching the timeline. Thus, without information shared between users, the linkage attack by users and real-time movement tracks are impossible.

False positives. Aside from real people misreporting their symptoms, applications that rely on diagnosed users' information are at risk of malicious false-positive claim. For location-based applications, these attacks are very effective. For a GPS based system, an attacker could spoof a series of GPS coordinates to an app. If no authentication measures are implemented, uploading such spoofed GPS data to the server can cause potential havoc across the system. This scenario, in retrospect, emphasizes the design of permission code in our recommendation. A user is allowed to register as infected only after being authorized with a permission code, which prevents false-positive claims.

Further considerations. To protect users' privacy against relay attacks, the *VenueTrace* solution can be updated by including blurred location information in broadcasting. As a relay attack will replay a VenueID in another remote place and thereby expanding the broadcast range and causing false-positive alarms, we can distinguish fake broadcasts by combining the location of broadcasters in the broadcast message. When the user receives a broadcast, the application can parse out the location of the broadcaster and compare it with the location of a receiver to filer those bounded by a distance, *e.g.* greater than 1 km. This allows eliminating such replay forgeries.

As the original intention of the *VenueTrace* solution is to prevent the back-end server from obtaining locations of broadcasters and users, we can only add vague location information to the broadcast information, such as adding a 1 km error, to prevent the relay attack while preserving users' privacy. However, this, to some extent, weakens the location privacy protections of our solutions.

6 RELATED WORK

Location tracking. Recent works have explored extracting location information from mobile apps [45, 48, 53, 55]. For instance, Xue *et al.* [54] presented a supervised machine learning methodology to localize users without any reverse engineering of the app. In our vetting, we considered such situation as a linkage attack by users, which extended the scope to also include tracking via Bluetooth. Although it may cost much to build up a large-scale Bluetooth broadcasting and receiving network, *e.g.* a honeycomb layout, to apply a movement tracking attack, it is still a considerable risk of privacy leakage.

Contact tracing tools analysis. Several recent works have focused on the evaluation and analysis of contact tracing applications. Just 10 days after TraceTogether launched, Cho *et al.* [26] presented a constructive discussion of potential modifications to encourage community efforts to develop solutions with stronger privacy protection and argued that privacy is a central feature of mobile contact tracing apps. One week later, Vaudenay [52] analyzed the DP3T solution and pointed out that some privacy protection measurements by DP3T may have the opposite affect. Li and Guo [39] conducted

a pioneering review of the centralized and decentralized solutions and identified the directions of research. Liu *et al.* [40] propose a zero-knowledge protocol to provide privacy-preserving contact tracing.

Our work. In contrast to other works, our research not only focuses on the analysis of one solution or the comparison between centralized and decentralized designs, but also conducts a security and privacy vetting on multiple state-of-the-practice approaches. Concretely, we aim to extract security and privacy guidance from various solutions and propose more practical protection of individual security and privacy.

7 CONCLUDING REMARKS

This paper has conducted a security analysis of 35 contact tracing applications and evaluated the privacy performance of 10 solutions. The results show that security risks remain; such as logging sensitive data, using deprecated cryptographic algorithms, and insufficient protections of components. Thus, we recommend that the reported vulnerabilities be patched as soon as possible, although we appreciate that developers may prioritize the speed of product release to counter the pandemic. That said, the majority of patches are straightforward. For example, 70% of developers still use insecure hash functions such as `SHA-1` and `MD5`, and log sensitive data in delivered apps. Further, to ensure security and remove potential vulnerabilities, code should be released for public review.

Our analysis has shown that protecting privacy is more challenging, particularly as this must be balanced against the urgency of the pandemic. To the best of our knowledge, there are no solutions that can protect users' privacy against all potential attacks. Besides the solutions that collect private information, the results of our privacy vetting indicates that most of the contact tracing apps are potentially vulnerable to malicious privacy attacks.

To overcome a number of these issues, we have proposed a privacy-preserving contact tracing design, termed *VenueTrace*. The proposed recommendation has a decentralized architecture in which no information is exchanged among users and no location and identifiable information will be exposed to the server. However, just as with other apps, it is impossible to address all potential risks, *e.g.* our solution is similarly vulnerable to the relay attack; a solution requires compromising the privacy of our approach. We hope our study can inform and aid the software industry to design, develop, and deploy more secure (and privacy-preserving) contact tracing apps whilst allowing citizens to use contact tracing apps with more confidence in the capability of the apps to protect their security and privacy.

REFERENCES

[1] [n. d.]. Android Root Detection Techniques. ([n. d.]). https://blog.netspi.com/android-root-detection-techniques/
[2] [n. d.]. COVIDSafe Android App - BLE Privacy Issues. ([n. d.]). https://docs.google.com/document/d/1u5a5ersKBH6eG362atALrzuXo3zuZ70qrGomWVEC27U/edit#
[3] [n. d.]. COVIDSafe Source Code. ([n. d.]). https://github.com/AU-COVIDSafe
[4] [n. d.]. CWE-330: Use of Insufficiently Random Values. ([n. d.]). https://cwe.mitre.org/data/definitions/330.html
[5] [n. d.]. Geoffrey Huntley's Twitter. ([n. d.]). https://twitter.com/GeoffreyHuntley/status/1258174433979625474?s=20
[6] [n. d.]. Logcat command-line tool. ([n. d.]). https://developer.android.com/studio/command-line/logcat

[7] [n. d.]. Man-in-the-middle attack. ([n. d.]). https://en.wikipedia.org/wiki/Man-in-the-middle_attack

[8] [n. d.]. Mobile-Security-Framework-MobSF. ([n. d.]). https://github.com/MobSF/Mobile-Security-Framework-MobSF

[9] [n. d.]. OWASP. ([n. d.]). https://owasp.org/www-project-mobile-top-10/

[10] [n. d.]. Privacy Issues Reported By Geoffrey Huntley. ([n. d.]). https://twitter.com/GeoffreyHuntley/status/1257540474048581632?s=20

[11] [n. d.]. Private Kit: Safe Paths; Privacy-by-Design COVID-19 Solutions using GPS+Bluetooth for Citizens and Public Health Officials. ([n. d.]). http://safepaths.mit.edu/

[12] [n. d.]. Protecting Against Linkage Attacks that Use âĂŸAnonymous DataâĂŹ. ([n. d.]). https://www.marklogic.com/blog/protecting-linkage-attacks-use-anonymous-data/

[13] [n. d.]. Remote Debugging WebViews. ([n. d.]). https://developers.google.com/web/tools/chrome-devtools/remote-debugging/webviews

[14] [n. d.]. Support Direct Boot mode. ([n. d.]). https://developer.android.com/training/articles/direct-boot

[15] [n. d.]. A 'travel log' of the times in South Korea: Mapping the movements of coronavirus carriers. ([n. d.]). https://www.washingtonpost.com/world/asia_pacific/coronavirus-south-korea-tracking-apps/2020/03/13/2bed568e-5fac-11ea-ac50-18701e14e06d_story.html

[16] [n. d.]. Understanding Multidex in Android. ([n. d.]). https://blog.mindorks.com/understanding-multidex-in-android

[17] [n. d.]. Using The SQLite Encryption Extension. ([n. d.]). https://sqlite.org/android/doc/trunk/www/see.wiki

[18] [n. d.]. Virustotal. ([n. d.]). https://www.virustotal.com/

[19] [n. d.]. Visual Studio Code. ([n. d.]). https://code.visualstudio.com/

[20] 2020. Coronavirus Desease-19. (2020). http://ncov.mohw.go.kr/,2020,accessed:2020-03-23

[21] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.

[22] Mehrdad Bahrini, Nina Wenig, Marcel Meissner, Karsten Sohr, and Rainer Malaka. 2019. HappyPermi: Presenting Critical Data Flows in Mobile Application to Raise User Security Awareness. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–6.

[23] Ashish Bhatia. 2019. Android Security: Don't leave WebView debugging enabled in production. (2019). https://dev.to/ashishb/android-security-don-t-leave-webview-debugging-enabled-in-production-5fo9

[24] Justin Chan, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob Sunshine, et al. 2020. PACT: Privacy Sensitive Protocols and Mechanisms for Mobile Contact Tracing. *arXiv preprint arXiv:2004.03544* (2020).

[25] Bill Chappell. 2020. Coronavirus: Sacramento County Gives Up On Automatic 14-Day Quarantines. (2020). https://www.npr.org/sections/health-shots/2020/03/10/813990993/coronavirus-sacramento-county-gives-up-on-automatic-14-day-quarantines,2020,accessed:2020-03-23

[26] Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. 2020. Contact tracing mobile apps for COVID-19: Privacy considerations and related trade-offs. *arXiv preprint arXiv:2003.11511* (2020).

[27] Chris Culnane and Kobi Leins. 2019. Misconceptions in Privacy Protection and Regulation. *Law in Context. A Socio-legal Journal* 36, 2 (2019), 1–12.

[28] Australia Department of Health. 2020. Coronavirus Australia app. (2020). https://www.health.gov.au/resources/apps-and-tools/coronavirus-australia-app,2020,accessed:2020-04-23

[29] Australia Department of Health. 2020. COVIDSafe. (2020). https://www.health.gov.au/resources/apps-and-tools/covidsafe-app,2020,accessed:2020-04-23

[30] Luca Ferretti, Chris Wymant, Michelle Kendall, Lele Zhao, Anel Nurtay, Lucie Abeler-Dörner, Michael Parker, David Bonsall, and Christophe Fraser. 2020. Quantifying SARS-CoV-2 transmission suggests epidemic control with digital contact tracing. *Science* (2020).

[31] Tzipora Halevi, Di Ma, Nitesh Saxena, and Tuo Xiang. 2012. Secure proximity detection for NFC devices based on ambient sensor data. In *European Symposium on Research in Computer Security*. Springer, 379–396.

[32] Zhiyun Qian Hang Zhang, Dongdong She. 2015. Android Root and its Providers: A Double-Edged Sword. (2015). https://dl.acm.org/doi/abs/10.1145/2810103.2813714

[33] Yangyu Hu, Haoyu Wang, Li Li, Yao Guo, Guoai Xu, and Ren He. 2019. Want to earn a few extra bucks? A first look at money-making apps. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 332–343.

[34] Prof Hubaux. 2020. *Decentralized Privacy-Preserving Proximity Tracing*. Ph.D. Dissertation. Fraunhofer HHI.

[35] Alvin Tan Chai Sheng Hau Lai Yongquan Janice Tan Tang Anh Quy Jason Bay, Joel Kek. 2020. BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders. *Government Technology Agency, Singapore* (2020). https://bluetrace.io/

[36] Siddique Latif, Muhammad Usman, Sanaullah Manzoor, Waleed Iqbal, Junaid Qadir, Gareth Tyson, Ignacio Castro, Adeel Razi, Maged N. Kamel Boulos, Adrian Weller, and Jon Crowcroft. 2020. Leveraging Data Science To Combat COVID-19: A Comprehensive Review. (4 2020). https://doi.org/10.36227/techrxiv.12212516.v1

[37] Kobi Leins, Chris Culnane, and Benjamin IP Rubinstein. 2020. Tracking, tracing, trust: Contemplating mitigating the impact of COVID-19 through technological interventions. *The Medical Journal of Australia* (2020), 1.

[38] Douglas J Leith and Stephen Farrell. 2020. Coronavirus Contact Tracing App Privacy: What Data Is Shared By The Singapore OpenTrace App? (2020).

[39] Jinfeng Li and Xinyi Guo. 2020. COVID-19 Contact-tracing Apps: A Survey on the Global Deployment and Challenges. *arXiv preprint arXiv:2005.03599* (2020).

[40] Joseph K Liu, Man Ho Au, Tsz Hon Yuen, Cong Zuo, Jiawei Wang, Amin Sakzad, Xiapu Luo, and Li Li. [n. d.]. Privacy-Preserving COVID-19 Contact Tracing App: A Zero-Knowledge Proof Approach. ([n. d.]).

[41] Tianming Liu, Haoyu Wang, Li Li, Xiapu Luo, Feng Dong, Yao Guo, Liu Wang, Tegawendé Bissyandé, and Jacques Klein. 2020. MadDroid: Characterizing and Detecting Devious Ad Contents for Android Apps. In *Proceedings of The Web Conference 2020*. 1715–1726.

[42] Israel Ministry of Health. 2020. Hamagen. (2020). https://govextra.gov.il/ministry-of-health/hamagen-app/,2020,accessed:2020-04-23

[43] World Health Organization. 2020. Operational considerations for case management of COVID-19 in health facility and community. (2020). https://apps.who.int/iris/bitstream/handle/10665/331492/WHO-2019-nCoV-HCF_operations-2020.1-eng.pdf

[44] Raymond Zhong Paul Mozur and Aaron Krolik. 2020. In Coronavirus Fight, China Gives Citizens a Color Code, With Red Flags. (2020). https://www.nytimes.com/2020/03/01/business/china-coronavirus-surveillance.html,2020,accessed:2020-05-07

[45] Iasonas Polakis, George Argyros, Theofilos Petsios, Suphannee Sivakorn, and Angelos D Keromytis. 2015. Where's Wally? Precise user discovery attacks in location proximity services. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 817–828.

[46] Ramesh Raskar, Isabel Schunemann, Rachel Barbar, Kristen Vilcans, Jim Gray, Praneeth Vepakomma, Suraj Kapa, Andrea Nuzzo, Rajiv Gupta, Alex Berke, et al. 2020. Apps gone rogue: Maintaining personal privacy in an epidemic. *arXiv preprint arXiv:2003.08567* (2020).

[47] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. 2014. A machine-learning approach for classifying and categorizing Android sources and sinks. In *NDSS*, Vol. 14. Citeseer, 1125.

[48] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. 2019. A billion open interfaces for Eve and Mallory: MitM, DoS, and tracking attacks on iOS and macOS through Apple Wireless Direct Link. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 37–54.

[49] Daniel Blank Jesse Colligan Rhys Fenwick Mike Hittle Mark Ingle Oliver Nash Victoria Nguyen James Petrie Jeff Schwaber Zsombor Szabo Akhil Veeraghanta Mikhail Voloshin Tina White Helen Xue Sydney Von Arx, Isaiah Becker-Mayer. 2020. Slowing the Spread of Infectious Diseases Using Crowdsourced Data. (2020). https://www.covid-watch.org/covid_watch_whitepaper.pdf,2020,accessed:2020-05-07

[50] Yvonne Taunton. 2020. Smartphone-based automated contact tracing: Is it possible to balance privacy, accuracy and security? (2020). https://www.uab.edu/news/research/item/11299-smartphone-based-automated-contact-tracing-is-it-possible-to-balance-privacy-accuracy-and-security,2020,accessed:2020-05-07

[51] The PEPP-PT team. 2020. PEPP-PT High Level Overview. (2020). https://github.com/pepp-pt/pepp-pt-documentation/blob/master/PEPP-PT-high-level-overview.pdf

[52] Serge Vaudenay. 2020. Analysis of DP3TBetween Scylla and Charybdis. (2020). https://eprint.iacr.org/2020/399.pdf

[53] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y Zhao. 2016. Defending against Sybil devices in crowdsourced mapping services. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 179–191.

[54] Minhui Xue, Cameron Ballard, Kelvin Liu, Carson Nemelka, Yanqiu Wu, Keith Ross, and Haifeng Qian. 2016. You can yak but you can't hide: Localizing anonymous social network users. In *Proceedings of the 2016 Internet Measurement Conference*. 25–31.

[55] Minhui Xue, Yong Liu, Keith W Ross, and Haifeng Qian. 2016. Thwarting location privacy protection in location-based social discovery services. *Security and Communication Networks* 9, 11 (2016), 1496–1508.

[56] Fengjiao Wang Hongxia Jin Hongxia Jin Yilin Shen, Fengjiao Wang. 2014. Defending against User Identity Linkage Attack across Multiple Online Social Network. (2014).

APPENDIX

A STATIC ANALYSIS

Code analysis. MobSF is a state-of-the-art and open source pen-testing, malware analysis and security vetting framework [22], which flags vulnerabilities in code.

For context, MobSF works in the following way. The de-compiled `AndroidManifest.xml` file is first parsed to extract essential information about the application, such as `Permission`, `Components`, `Intents`. Then, the system assess requested permissions by the application and examine whether all `Components` (*e.g.* `Service`, `Receiver`, `Activity`, `Provider`) are protected by at least one permission explicitly requested in manifest file. Other attribute configurations, such as the `allowBackup`, `debuggable`, and `networkSecurityConfig` flags, will also be checked.

The `class` files are subsequently parsed via a Sensitive Data Match module, which utilizes keyword matching, *e.g.* "password" and "secret". The Method Extraction module matches methods in `class` files with pre-defined rules to extract vulnerable methods. For example, if a method contains the keyword `.hashCode()`, it will be considered as using Java Hash Code, a weak hash function that should not be used in a secure cryptography implementation. However, as a weakness could be defined in third-party APIs, the vulnerable method may never be executed during run-time. To address this, the Determining vulnerable Calls module will vet whether a vulnerable method is actually called and assess whether the sensitive data is accessed. The system will record all the vulnerabilities listed in the *Manifest Weaknesses* and *Vulnerabilities* categories in Table 3. Further, the trackers in apps, *e.g.* `Google Firebase Analytics`, `Facebook Analytics`, and `Microsoft Appcenter Analytics` are detected by the Tracker Detection module and will be recorded in the *Privacy Leaks* category in Table 3.

The vetted vulnerabilities include SQL injection, IP address disclosure, hard-coded encryption keys, improper encryption, use of insufficiently random values (CWE 330) [4], insecure hash functions, and remote WebView debugging is enabled. We detail the manual inspections adopted subsequently in the vetting process in Appendix B and limitations to the code analysis vetting methods in Appendix D.

Data flow analysis. We conduct a data flow analysis using Flow-Droid [21] to screen out high risk privacy leaks. Such data flow analysis extracts the paths from data sources to sinks, and the statements transmitting the data outside of the application. We use the sources and sinks inferred by SuSi project [47] which defines sources as calls to resource methods, *e.g.* `getLatitude()` and `database.Cursor.getString()`, while sinks are methods that may leak sources, *e.g.* `Log.e()` and `Bundle.putAll()`.

FlowDroid searches the application for lifecycle and callback methods and then generates a call graph. Starting at the detected sources, the analysis tracks taints by traversing the call graph. If private data flows from a source to sink, it indicates that there is a risky privacy leak path. To remove false-positives, we conduct a backward flow analysis. If the vulnerable code is reachable, we determine it is a valid privacy leak. For example, if we find there is sensitive data that flows into a sink (*e.g.* Bundle, Log output, SMS) unauthorized users can access, we will trace it backwards to its source and confirm whether the source is reachable. If reachable, we consider it as a privacy leakage. Similar to the code analysis phase, we also conduct a manual inspection described in Appendix C. We describe the limitations to the vetting method in Appendix D.

B CODE ANALYSIS: MANUAL INSPECTIONS

To increase the accuracy of the vetting results, we manually verified the testing results of MobSF. First, considering that MobSF mainly relies on keywords and sentences matching in APIs, we check the rules defined in MobSF's source code. Then, we collect rules with weak keywords defined. For example, if a rule only uses `Log.v` or `System.out.print` to find "sensitive data logging", without checking whether the data is sensitive, we consider this is a weak rule. We automated the extraction of all Java files which are identified by these weak rules for manual inspection. Finally, we removed the false-positive cases from the testing results.

C DATA FLOW ANALYSIS: MANUAL INSPECTIONS

For data flow taint analysis, we manually review the FlowDroid results of all 35 apps. Since all APK files have been decompiled by MobSF, sink paths in XML reports generated by FlowDroid are closely contrasted with decompiled source codes in order to identify false-positive cases. First, sink paths are picked out to analyze the possibility of potential leakage. If any of these paths are suspected to be false-positive cases, we load source codes decompiled by MobSF in Visual Studio Code [19] and use its global search feature to find invoked methods mentioned in suspected sink paths. Finally, after analyzing the logic from source code, false-positive cases are confirmed.

D THREATS TO VALIDITY

Potential limitations to our methodology. Considering that both the core mechanisms of MobSF and FlowDroid heavily rely on keywords matching, a potential cause of false negatives is largely due to the scope of keywords. Concretely, in MobSF, there may exist vulnerabilities not defined in analysis rules. Similarly, in our data flow vetting, as we utilize the sources and sinks extracted by SuSi project, there may exist sensitive leakage that does not match any sources or sinks or that is not detected. We aim to improve the false negative by updating the rules and keywords database of MobSF and FlowDroid in future avenue. Currently, our vetting focuses more on the identified vulnerabilities and privacy leakage paths.