Security and Privacy Analysis of Android Family Locator Apps

Khalid Alkhattabi
Colorado School of Mines
Golden, Colorado
kalkhattabi@mines.edu

Ahmed Alshehri
Colorado School of Mines
Golden, Colorado
alshehri@mines.edu

Chuan Yue
Colorado School of Mines
Golden, Colorado
chuanyue@mines.edu

ABSTRACT

Families are increasingly using Family Locator (FL) apps for convenience and safety purposes. Such FL apps often collect a lot of sensitive information, such as user location and contacts, to improve their usability and functionality. However, it is not clear if they provide strong protections to the collected sensitive information or not. This paper presents the findings on the first security and privacy analysis of FL apps. We select 41 FL apps from the Google Play store. We first analyze the permissions requested by the FL apps to understand the types of sensitive information they would collect. Then, we analyze the network traffic and local storage of these apps to identify potentially sensitive information leakage. Our analysis demonstrates that significant security and privacy vulnerabilities exist among FL apps. Specifically, 80.4% of the 41 FL apps leak sensitive information or join codes in plaintext. A join code would allow an attacker to join a family's group to perform a wide range of malicious activities. Meanwhile, we found that 15.1% of the 33 apps leak sensitive information from their back-end servers due to authentication and authorization vulnerabilities. We provide suggestions to users and developers of FL apps to improve security and privacy. We responsibly disclosed our findings to the developers of the 33 vulnerable apps. Nine of the developers confirmed our findings and showed interest in addressing them in their next updates. The feedback from our responsible disclosures shows that our analysis makes an impact on the security and privacy of FL apps.

CCS CONCEPTS

• **Security and privacy** → **Social network security and privacy**; **Access control**;

KEYWORDS

Android, Family Locator App, Security, Privacy

ACM Reference Format:
Khalid Alkhattabi, Ahmed Alshehri, and Chuan Yue. 2020. Security and Privacy Analysis of Android Family Locator Apps. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3381991.3395612

1 INTRODUCTION

Family Locators (FL) apps allow sharing location and other information of users in a group (family) for convenience and safety purposes. Some FL apps also allow remote activation of the microphone or camera on another family member's device. For example, parents can be kept apprised of their children's surroundings in addition to their actual location. FL apps also have other features like the ability to designate an area as a danger zone. With this feature, parents receive a notification whenever a child or some other family member enters a designated danger zone. Other features include the ability to monitor battery status and save location history. Currently, FL apps are not used exclusively for normal family communication, but can also be used to monitor other vulnerable individuals, such as people with mental or physical disabilities. Such individuals might require special oversight, and FL apps can greatly support their safety and well-being.

When it comes to using technology that collects data about individuals, privacy concerns arise. In the United States and Europe, there are some privacy regulations to restrict the use of personal data. One of the privacy regulations is Children's Online Privacy Protection (COPPA). COPPA is to regulate how mobile apps or websites collect online information about kids under 13 years [7]. COPPA states that mobile apps should get parents' consent before they start collecting, using, or disclosing personal information about their kids. This includes geolocation, address, and multimedia data. COPPA also states that mobile apps should maintain the confidentiality and integrity of children's information. If mobile apps violate any part of COPPA Federal Trade Commission (FTC) might issue a fine as a penalty against the developers for violation. FL apps collect a lot of data about kids, and developers should pay close attention to not violate such regulations.

FL apps are unlike traditional apps. Traditional apps collect sensitive information about individuals. However, FL apps collect sensitive information about individuals and their family members. If an FL app stores or transfers sensitive information such as personally identifiable information (PII) in an insecure format, the whole family will be exposed. To illustrate, if criminals are able to access location information about the whole family, their house can be targeted while the family is away. Moreover, FL apps use join codes (secret codes) to invite people into circles (groups) for the purpose of sharing location and interacting. Anyone with a join code can enter the family circle and access the location of all the members as well as read all messages in the group's chat.

Recent articles in the news have shown that an FL app suffers from issues related to security and privacy. It reported that one particular FL app leaked a massive amount of sensitive information ranging in type from personal data to corporate secrets of more than 238,000 users [30]. The main cause of the data breach was that a vulnerability was exploited in the app's server. Additionally, FL apps might leak sensitive information at rest (device storage) or in transfer over the network. FL join codes and other personal data can be exposed to cybercriminals if appropriate protection measurements are not in place. The attacker in this scenario may employ either a local or network attack to stealthily obtain or interfere with communications of the group. Therefore, FL apps need to ensure the confidentiality and integrity of user data, particularly when information, such as a join code, is being transmitted or stored.

This research aims to analyze the security and privacy of FL apps on the Android platform. Specifically, it analyzes certain critical channels of the apps that might be exploited. We monitor and analyze the communication between mobile devices and the corresponding back-end servers to determine whether user data is being transferred securely. Also, we analyze the use of local storage to determine whether user data is stored in an adequately secure fashion. The research question is: Do family locator (FL) apps protect family data?

A number of related studies [5, 6, 10–12, 18] have investigated the security and privacy protections in banking, mobile health (m-health), and dating apps. However, research in the category of FL apps is lacking, and the security and privacy features of FL apps have yet to be analyzed. To the best of our knowledge, our study is the first to examine FL apps from the security and privacy perspectives.

To answer our research question, a static analysis approach like in [2, 3, 8] could be used to trace sensitive information in the FL apps. However, the main drawback of static analysis is that it is susceptible to false positives. In addition, code obfuscation techniques, which are commonly used in apps, affect the accuracy of static analysis. As a result, it is hard to analyze certain vulnerabilities such as join code leakage simply by code analysis.

In this work, we use a dynamic analysis approach to overcome the drawbacks of static analysis. We take a semi-automated approach to investigate the security and privacy issues in FL apps. First, we identify candidate apps by crawling Google Play with the keyword phrase "family locator" which allows us to identify 41 candidate FL apps. These apps have millions of installs. Then, we extract a list of permissions from the Android Manifest file of each app to understand what types of sensitive data they gather. Subsequently, we install each app on two devices to simulate family member interactions. In this step, it is necessary to manually interact with the app to establish an account, create a circle, and invite group members into that circle. Lastly, we leverage the network traffic and local storage of each app to evaluate and analyze their security and privacy functions.

The results of our analysis show that significant security and privacy issues exist in most of those FL apps. We found that 33 apps (80.4%) did not provide proper protection to keep sensitive information secure over network communications or in local storage. Among the 33 vulnerable apps, 24 apps (72.7%) leak users' location data in insecure forms either over HTTP or in local storage. We also found that nine (27.3%) of the 33 vulnerable apps leak join codes in insecure forms either over network traffic or in local storage, meaning that an attacker might use these join codes to join a family group to carry out malicious attacks. We also discovered three (9.0%) of the 33 vulnerable apps make use of SD card (external storage) to store voice chats and group members' information in insecure forms; any app with an *external storage permission* can access the information stored on the SD card by other apps. In addition, we discovered some of the FL apps leak sensitive information from their corresponding servers due to an absence of authentication and authorization mechanisms; this vulnerability was identified in five (15.1%) of the 33 vulnerable apps.

We reached out to the developers of the vulnerable 33 FL apps to disclose our findings responsibly. We received 14 responses so far. Nine of the developers confirmed our findings and showed interest in addressing them in their next updates.

We provide some suggestions to both developers and users of FL apps to protect against data leakages. For developers, implementing secure storage and transit of the data can fix most of the vulnerabilities found in this study. For users, they need to be aware of the potential security and privacy limitations of FL apps.

In summary, our paper makes the following contributions:

- We perform the first security and privacy analysis of FL apps which touch many types of sensitive data of users.
- We design a semi-automated framework that analyzes network traffic and local storage of FL apps to discover security and privacy vulnerabilities.
- We use our framework to analyze 41 FL apps, and find that 80.4% of them do not provide proper protection of user data in local storage or over network traffic, 72.7% of them leak users location data, and 15.1% of them lack authentication and authorization checks.
- We responsibly disclose our findings to the developers of FL apps, and provide suggestions to both developers and users.

The rest of the paper is organized as follows: Section 2 provides background and discusses related research. Section 3 presents the thread model. Section 4 describes the methodology we deploy for the FL app analysis. Analysis results are in Section 5, and case study is in Section 6. Section 7 presents discussion about FL apps security issues and potential mitigation. Section 8 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 Background

2.1.1 Family Locator Apps. FL apps are extremely useful for keeping families informed of the whereabouts of their children, other family members, or even friends. Figure 1 presents an overview of the process of installing and using a typical FL app, which includes establishing an account, creating a circle, and inviting individuals to join the circle/group using the FL app. Once the app is installed, the user must create an account to use the app. Registration steps are simple. FL apps only need a small amount of information, such as an email address, phone number, password, and name. Some apps ask for specific permissions before allowing the user to complete the registration form (e.g., READ_PHONE_STATE, READ_CONTACTS). However, some FL apps do not ask users to create an account. For these apps, the user only needs a join code to join the circle to start chatting with family members and accessing the group location data.

After completing the registration, the user might need to grant a set of permissions for an app. For example, some FL apps ask for location permission to access the user's location. Once these permissions have been granted, the user will then be able to create

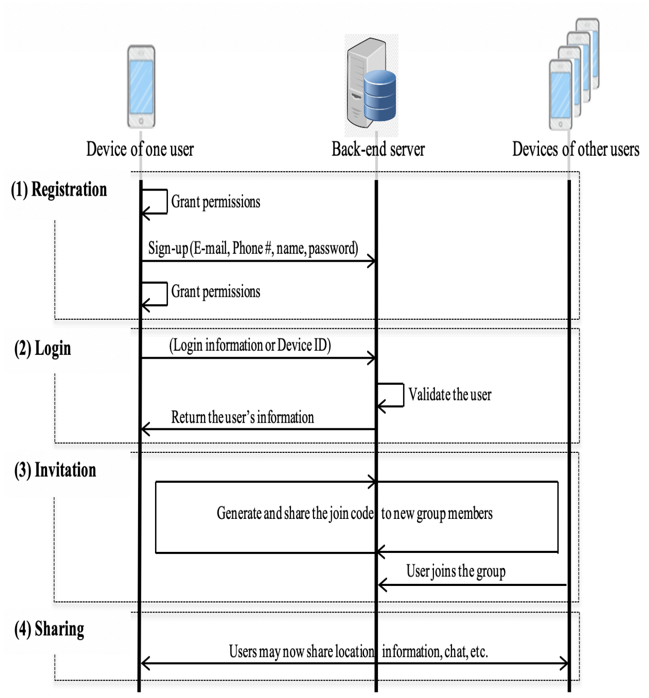
Device of one user
Back-end server
Devices of other users
(1) Registration
Grant permissions
Sign-up (E-mail, Phone #, name, password)
Grant permissions
(2) Login
(Login information or Device ID)
Validate the user
Return the user's information
(3) Invitation
Generate and share the join code to new group members
User joins the group
(4) Sharing
Users may now share location information, chat, etc.

Figure 1: Family locator app overview.

a new circle that others can be invited to join, meaning that the user can share location, communicate, or participate in other FL app activities.

To invite group members to a circle, the user needs to generate a join code and share it with them. The distribution of the join code may be achieved in several ways, such as through app notification messages, email, text messages, or social media. The join code (or secret code) can be generated by the user or, more often, by the app itself. A circle may have a fixed join code for all invitations or may set a specific join code for each invitation. Some apps set a lifetime on a join code ranging between three hours and seven days based on our analysis. The lifetime setting is to limit how long the join code can be used to enter a circle after an invitation has been sent. To join a circle, a group member uses the join code received from the circle owner; if the join code is valid, the user will be able to successfully join the circle and start using the features provided by the FL app with all group members in that circle.

Several features are commonly available with FL apps. Indeed, location sharing is the primary one. Location sharing provides the main data for safety features of group members. This is the reason why most of FL users find this feature essential. In addition, FL apps can be used by friends or business partners to agree on a place for a meeting using the Google Maps pin feature. Other available features include messaging, sending out an emergency notification (e.g., SOS or alarm), sharing events/activities, sharing lists (e.g., grocery or other shopping lists), checking battery life, disseminating meeting agendas, and providing information on location history.

Obviously, FL apps have a rich environment of sensitive information for the attackers. An attacker might seek out such information to carry out fraudulent activities. FL apps obtain a lot of sensitive information about group members, families, and friends. Unlike FL apps, traditional apps typically obtain information from a single user. Consequently, the harm on traditional apps might affect only one person while that on FL apps might harm the whole family.

The various potential vulnerabilities of FL apps can cause a leakage of sensitive information to an attacker. For example, if someone with malicious intent can intercept a join code, that person can join a circle and obtain a great deal of sensitive information (e.g., location, credit card numbers, PINs, or others). Therefore, to protect such confidential information, these apps should encrypt join codes during transfer and storage. Without such security measures, all users may be at risk. FL app developers should set security goals to maintain the confidentiality of user data in transit and at rest.

2.1.2 Data Leakage Channels. Communication channels are used to exchange data between client-server systems. FL apps need to transfer data to/from remote servers as part of their standard operations. Thus, data transmission must employ cryptographic mechanisms to prevent unauthorized disclosure of information. There are many security libraries like OpenSSL [16]. Simply, apps can leverage open source security libraries to protect their users. In evaluating the security and privacy of an FL app, it is essential to analyze the network traffic to identify areas where data security and privacy might be violated.

FL apps also need to store data on local storage for their operation. Such data may be leaked while being stored, processed, or at rest [23]. Certain types of malicious attackers seek out these types of vulnerabilities for data interception/alteration attacks to obtain or alter sensitive information. For example, an attacker may perform a data alteration attack to make modifications to data that can render it invalid or even being destroyed. Additionally, other advanced attacks may be possible, like replay attacks. Therefore, it is vital that FL apps store data in an encrypted form in local storage to avoid such vulnerability to these types of attacks.

It is clear by now that an attacker can take advantage of a lack of security in communication and storage channels to obtain PII data about users of FL apps. A data breach in either of these channels affects the confidentiality and integrity of the data. Thus, in this paper, we leverage communication and storage channels to detect any potential data breach in FL apps.

2.2 Related Work

Some researchers have analyzed security and privacy in dating, m-health, banking, and other categories of Android apps. Despite the importance in terms of the nature of data, no study has focused on FL apps.

Hu et al. analyzed the mobile ecosystem of Android dating apps and found several security vulnerabilities [12]. They performed a systematic analysis of fraudulent activities in dating apps. Mainly, they examined user behaviors, irrelevant messages, premium services, and user comments to detect malicious activities in dating applications. They found that many user accounts are fake or chatbots. In addition, the authors found that some fraudulent dating apps have fake user reviews and ratings in the markets. They found some apps luring users to buy premium services to continue chatting. Another study done by Kim F. M. provides a forensic analysis of nine popular dating apps. Users' actions were simulated in the apps. Authors then inspected internal and external storage, and network traffic [6]. They evaluated five sensitive kinds of data: messages, images, locations, email addresses, and authentication data. Among the nine apps, only one app stored messages in an encrypted form. They also found that only one app did not leak location data over the network or in local database. In a similar fashion, Shetty et al. demonstrated that many dating apps were vulnerable to a man-in-the-middle attack [24]. They demonstrated how an attacker could succeed in intercepting and obtaining personal and private information in most of the examined applications.

The security and privacy of m-health apps have been analyzed in [4, 11, 18]. The m-health apps have a PII-rich environment where confidentiality, integrity and privacy of users must be ensured and maintained. In [18], the authors performed both static and dynamic analysis of the selected m-health apps, along with tailored testing of each app's functionalities. This research also presented an analysis of security and privacy concerns in m-health apps through a long-term evaluation, monitoring, and assessment of the quality of all communication channels within the apps examined. A similar work in [11] studied 160 m-health apps offered by Google Play and formulated a list of seven attacks that should be taken into consideration when evaluating app security and privacy. These attacks targeted sensitive information on the network, third-party services, Bluetooth, logging, SD card storage, exported components, and side channels. Another work was done on m-health mobile apps by [4]. It analyzed m-health apps from two perspectives. First, the authors analyze EU and US data protection regulations and requirements for health data. Second, they analyzed a fitness app as a case study, showing that it did not protect personal data.

Banking apps have been inspected by researchers for their security and privacy. For example, the authors of [5] analyzed seven Android m-banking apps in Thailand and found many security-related issues. The apps did not encrypt user data and also were vulnerable to the repackaging attack, which would put users' data at risk. Fahl et al. reviewed several of the most popular free apps and looked at the current state of SSL/TLS to detect potential man-in-the-middle (MITM) attacks [10]. The study explored a variety of categories, including banking. Similar work was conducted in [14]. The authors analyzed 34 banking apps on the iOS platform. The authors investigated how user inputs can be intercepted and how jailbreak detection mechanisms could be avoided in these apps. They leveraged the Cydia Substrate platform to intercept sensitive user inputs. They were able to intercept users' inputs on 15 banking apps successfully. The remaining 18 banking apps have Jailbreak detection, which blocked the interception of users' inputs. Furthermore, they edited system functions and altered return values to make the mobile device appear like a non-jailbreak device for bypassing jailbreak detection. The result showed that 32 out of 34 (94%) banking apps were vulnerable to the input interception attack, even if banking apps had jailbreak detection mechanisms.

Our work differs from previous projects [6, 11, 18, 24] in terms of the focus of the analysis and the methodology used. First, our study conducts a comprehensive evaluation of the security and privacy of FL apps, which have different functionalities and sensitive data than dating, banking, and m-health apps. For example, unlike dating, banking, and m-health apps, FL apps rely on a join code; if the apps fail to protect this secret code, the whole system is vulnerable, and sensitive information may be leaked. Also, the potential damage in FL apps can be wider than other categories. For example, FL apps normally have more than one user, so the harm can affect a whole family instead of one person as in other categories. Second, our methodology covers the data both in transit and rest. In addition, we evaluate the configurations of the FL apps' back-end servers to determine whether they support transport security mechanisms. We also examined them for potential data leakage vulnerabilities like the absence of authentication and authorization mechanisms. In general, we study the security vulnerabilities and design weaknesses in FL apps in terms of confidentiality and integrity.

3 THREAT MODEL

Two main attacks are considered in the FL app threat model: network and local storage attacks. Figure 2 illustrates the threat model for our security and privacy analysis of FL apps.

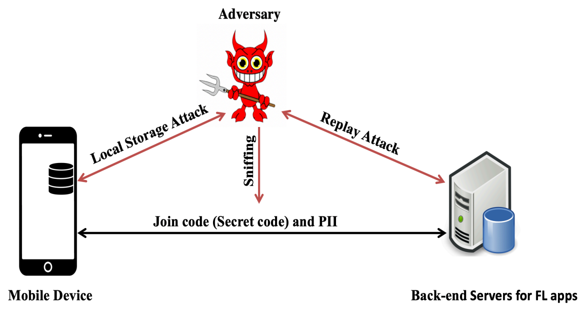

Figure 2: Threat model for security and privacy analysis of Android FL apps: network and local storage attacks.

Network attacks have two scenarios. In the first scenario, an attacker can intercept and obtain sensitive data and secrets from insecure communications. The goal for the attacker in this scenario is to capture sensitive data as they travel between FL apps and their corresponding servers. An attacker who can intercept join codes will be able to join the family circle. Upon joining, the attacker then has a wide range of capabilities, such as knowing the location of everyone in the circle and accessing family messages. Family messages may contain important secrets like door PIN codes or credit card numbers. The attacker can change the content of the message to achieve harmful goals if traffic is not encrypted like persuading some family members to go to a dangerous location. In the second scenario, the attacker looks for patterns in network traffic to reveal sensitive information from mobile back-end servers. The attacker first needs an initial understanding of what kind of information is required by the mobile back-end server to return user data. For example, by knowing that some FL apps use certain packets to retrieve data from their corresponding servers, attackers can reuse the same packets to have the same response. This attack is similar to the replay attack. Network attacks can be mitigated by encrypting traffic to prevent sniffing, and deploying a secure authentication mechanism at the server side to prevent replay attacks.

The Android architecture allows apps to store their data in two kinds of data storage: internal and external storage (SD card) [28]. In this paper, we refer to the internal and external storage together as local storage. The internal storage is used to keep the app's private data on the device mainly for performance reasons. It is located on the "data/data" path, and it includes the shared preferences and the database directories.

The second type is the external storage which is a shared space. All apps can store data on the external storage. Specifically, any app that has a *read_external_storage* permission can read the data stored on external storage, meaning that an attacker does not need a rooted device to access data stored by other apps on the external storage. An attacker or malicious app only needs to have *read_external_storage* permission. Therefore, sensitive data stored on external storage can be compromised via a covert channel attack [21], which exploits the feature that allows transferring sensitive information between apps. Android developer documentation states that storing private data on external storage is not recommended [28]. Data should always be stored in an encrypted form, so that attackers can not easily compromise it [32].

It is worth noting that we do not consider whether the FL apps are malicious apps. So, if owners of the apps want to compromise user data, they can easily do that. Other types of attacks, such as repackaging attacks, DDoS (Distributed Denial-of-Service) attacks, and attacks that target API vulnerabilities, are also not considered. In addition, we do not address leaks that occurred over encrypted channels, such as via SSL connections.

4 METHODOLOGY

4.1 Dataset

Our raw dataset contains 41 FL apps collected from the official Google Play store. We use a keyword matching method to find apps in the FL category. FL apps are listed in a variety of different categories in the store. Thus, we use the keyword "family locator" to identify as many as possible of FL apps for our study.

We also crawl the apps' descriptions, including the app name, publisher, developer's information, app version, number of downloads, and users' comments. Table 1 lists the 41 FL apps package names and their popularity in the Google play store. The majority of the FL apps in our dataset have more than 100,000 downloads, and five apps have more than 1,000,000 downloads.

4.2 Framework

Figure 3 shows the framework that we use to analyze the security and privacy of FL apps. It has three main components: App Permissions Extraction, Network Traffic Analysis, and Local Storage Analysis.

The App Permissions Extraction component is used to identify the most sensitive permissions requested by FL apps. By identifying and analyzing the sensitive permissions in these apps, we could understand what kinds of information leakage can potentially happen on both network traffic and local storage. Most of the FL apps request access to location, storage, and contact information. This analysis gives us insights into the type of sensitive information that can be leaked to unauthorized parties over leakage channels. We describe our findings in detail in Section 5.1.

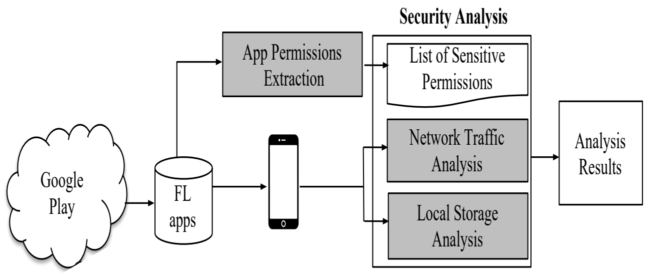

Figure 3: Methodology for analysis of Android FL apps.

In the Network Traffic Analysis component, we capture all traffic traces on an Android device for each app. Network traffic traces can be used to detect leakage of sensitive information over HTTP connections. If we find sensitive data in the traffic traces in plaintext, we raise concerns about app security.

For analyzing local storage, we first simulate functionalities of FL apps. Second, we record all the data that entered by the users. In addition, we record all permissions requested by the app. Third, we analyze the stored data on local storage if there are any security or privacy concerns.

4.3 Experimental Setup

Normally, FL apps are used by more than one user, e.g., a parent and their child. In order for us to have a good analysis, we need to install each app on two devices. We use the Android Debug Bridge (ADB) command-line to perform the installation process. Both Android devices have Android version 8, *which is known as Android Oreo.*

In order to launch an app, we extract the Launchable Activity from the Manifest file. Then, we pass the *launchable activity* to the ADB shell with the activity manager (AM) tool. We run each app for 20 minutes before uninstalling it. We think 20 minutes are enough to examine the main functionalities of each FL app. During the 20 minutes, we follow a predefined procedure for each app as follows: 1) granting all the required permissions; 2) creating an account for each app; 3) creating a family circle; 4) inviting group members; 5) sending chat messages.

After the predefined procedure above is completed, we use the Monkey tool to explore the app on the two devices (each device independently) [25]. The Monkey tool performs stress testing on Android apps using some random UI events. We use the Monkey tool to send 500 random events to the app such as clicking, touching, pressing home buttons, starting calls, and entering random texts in order to trigger all the possible functionalities of the app.

In Network Traffic Analysis, we use tcpdump for capturing traffic [26]. Traffic traces are separated on an Android device for each app. Network traffic traces then can be used to detect any leakage of user information over HTTP connections. Ideally, all traffic should be by default encrypted. In this analysis, any unencrypted traffic by any FL app raises a flag for concerns. Then if we find sensitive data in the unencrypted traffic, we label this app as vulnerable to sensitive data leakage.

Table 1: List of all FL apps in our dataset.

Index	Package name	Number of downloads
1	com.life360.android.safetymapd	50,000,000+
2	com.fsp.android.friendlocator	10,000,000+
3	com.foursquare.robin	10,000,000+
4	com.zoemob.gpstracking	5,000,000+
5	com.glympse.android.glympse	5,000,000+
6	app.gpsme	1,000,000+
7	app.alpify	1,000,000+
8	com.geozilla.family	1,000,000+
9	com.isharing.isharing	1,000,000+
10	com.sow.gpstrackerpro	1,000,000+
11	com.mspy.lite	1,000,000+
12	com.wavemarket.finder.mobile	1,000,000+
13	ar.com.megaingenieria.emobi.locator	1,000,000+
14	family.tracker.my	1,000,000+
15	net.familo.android	1,000,000+
16	iplay.mobiletracker.location.livemap	1,000,000+
17	mg.locations.track5	1,000,000+
18	net.cybrook.trackview	1,000,000+
19	com.family.locator.familylocator.app .new.navigacijos.mb	1,000,000+
20	com.sygic.familywhere.android	1,000,000+
21	com.familyhawk.app	100,000+
22	com.familywall	100,000+
23	city.russ.alltrackerfamily	100,000+
24	com.familyprotection	50,000+
25	locator24.com.main	50,000+
26	net.tifamily.android	50,000+
27	com.ilocatemobile.navigation	50,000+
28	com.capsa.my.family.locator	50,000+
29	com.testlab.family360	10,000+
30	kinkin.net	10,000+
31	com.gpswox.android_projects .familylocator	10,000+
32	com.ffinder.android	10,000+
33	com.fibercode.familytracker	10,000+
34	com.guerrillarocket.kidsmap	10,000+
35	com.placeter.familytracker	10,000+
36	com.maaya.myplace.locationshare .sharelocation.shareplace.mylocation	5,000+
37	com.friendlinks.mobile	1,000+
38	com.startappnow.familylocatoren	1,000+
39	com.haroonstudios.familytracklocator	500+
40	com.habron.habroncommunity	100+
41	com.panther.familyloco	100+

During the testing of the apps, we record any sensitive information that can be entered by users, such as usernames, passwords, join codes, contact lists, and other group members' information. We feed these items to our framework. Our framework utilizes a python script to parse and analyze network traffic and local storage to examine if PII and secrets have been leaked in plaintext. The network traffic and local storage files collected from the two devices are used to analyze the security and privacy of the FL apps. Every app is tested for 20 minutes, and we then remove the app from both phones.

4.4 Security and Privacy Analysis

4.4.1 Analysis of Information leakage. First, traffic traces are used to identify data leakages of PII over HTTP. For our analysis, we focus on the sensitive information identified from the permissions analysis, and also the credential information that we collect during the manual testing, such as join codes. Some of this information is required for FL app functionality. However, sensitive data and secrets should never travel across the network in plaintext. We use direct pattern matching in network traffic to determine the leakage of sensitive information, such as location, join codes, contact information, passwords, device ID, email addresses.

Furthermore, we extract all data stored by each app on Android internal storage, specifically, on the local databases. Then, we parse and display the data to determine whether user data are stored securely in internal storage. We use string pattern matching to detect sensitive information in plaintext in local databases.

For external storage analysis, we extract the data from the SD card folder before uninstalling each FL app during the testing phase. We use the SD card folder for each FL app to find evidence of storing sensitive information on external storage in an unencrypted format. The SD card (external storage) is normally used to store multimedia files; however, some FL apps use the external storage to store sensitive information about FL app users.

4.4.2 Analysis of Back-end Servers. We used the network traffic analysis tool *tcpdump*, and the reverse engineering tool *apktool* to extract all URLs within each FL app [26, 31]. With the network traffic tool, we analyze all HTTP requests to understand what kind of information is required for servers to retrieve user data. We notice that a number of back-end servers of the FL apps rely on easily guessable information to return user information, such as the userID. The HTTP requests gathered from this procedure are used to test if it is possible to bypass the authentication scheme of the back-end server. To verify, we use a Chrome browser and resend the same HTTP requests that are extracted from the network traffic. If we can obtain user information from the server without an error message, we determined that a misconfiguration of user authentication exists. In this scenario, an attacker can retrieve many types of sensitive information about FL app users from the corresponding server by just guessing their userID. The absence of an authentication mechanism is a serious issue. Normally, security vulnerabilities may require some technical background like code analysis to carry harmful attacks. However, these kinds of vulnerabilities make it trivial for attackers to make a huge damage to FL app users as they can just guess user IDs to retrieve sensitive information.

Additionally, we use Exerciser Monkey that generates random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events [25]. However, Exerciser Monkey does not cover all code branches. This is considered one of its limitations [22]. As a result, we use *apktool* to extract all the URLs in the source code [31]. The rationale behind this is to verify whether FL apps' back-end servers implement an authentication mechanism. Also, we need to extract all HTTPS URLs in the FL app source code

Table 2: List of all Android permissions that we found in 41 FL apps.

Index	Permission	Number of Apps	Index	Permission	Number of Apps
1	ACCESS_NETWORK_STATE	41 (100%)	22	USE_CREDENTIALS	4 (9.8%)
2	INTERNET	41 (100%)	23	**WRITE_CONTACTS**	4 (9.8%)
3	**LOCATION**	41 (100%)	24	BLUETOOTH	3 (7.3%)
4	C2DM.PERMISSION.RECEIVE	35 (85.4%)	25	REQUEST_INSTALL_PACKAGES	3 (7.3%)
5	**WRITE_EXTERNAL_STORAGE**	35 (85.4%)	26	ACCESS_COARSE_UPDATES	2 (4.9%)
6	ACCESS_WIFI_STATE	30 (73.2%)	27	BLUETOOTH_ADMIN	2 (4.9%)
7	**READ_CONTACTS**	24 (58.5%)	28	PACKAGE_USAGE_STATS	2 (4.9%)
8	**READ_EXTERNAL_STORAGE**	23 (56.0%)	29	READ_OWNER_DATA	2 (4.9%)
9	**READ_PHONE_STATE**	22 (53.7%)	30	FLASHLIGHT	2 (4.9%)
10	**CAMERA**	21 (51.2%)	31	**ACCESS_SUPERUSER**	1 (2.4%)
11	C2D_MESSAGE	19 (46.3%)	32	BIND_ACCESSIBILITY_SERVICE	1 (2.4%)
12	GMS.PERMISSION.ACTIVITY _RECOGNITION	16 (39.0%)	33	CAPTURE_AUDIO_OUTPUT	1 (2.4%)
13	**GET_ACCOUNTS**	12 (29.3%)	34	READ_LOGS	1 (2.4%)
14	**CALL_PHONE**	11 (26.8%)	35	MODIFY_PHONE_STATE	1 (2.4%)
15	GET_TASKS	10 (24.4%)	36	READ_SYNC_STATS	1 (2.4%)
16	**RECORD_AUDIO**	8 (19.5%)	37	**SDCARD_WRITE**	1 (2.4%)
17	REQUEST_IGNORE _BATTERY_OPTIMIZATIONS	7 (17.1%)	38	USE_FINGERPRINT	1 (2.4%)
18	MODIFY_AUDIO_SETTINGS	6 (14.6%)	39	**WRITE_CALENDAR**	1 (2.4%)
19	**READ_PROFILE**	6 (14.6%)	40	WRITE_SETTINGS	1 (2.4%)
20	BATTERY_STATS	4 (9.8%)	41	COM.BROWSER.PERMISSION. READ_HISTORY_BOOKMARKS	1 (2.4%)
21	**READ_CALENDAR**	4 (9.8%)	42	INTERACT_ACROSS_USERS_FULL	1 (2.4%)

to test if they encrypt their traffic or not by simply checking for the existence of SSL/TLS. We also leverage *Qualys SSL Test* [20]. This tool helps us measure SSL/TLS configurations at the server-side. It also determines whether a server has an invalid certificate, an invalid configuration, an unknown CA, or other security issues. In addition, Qualys tests for SSL/TSL configurations if they have weak encryption suites, a vulnerability to the Heartbleed bug [9], or a vulnerability to POODLE (Padding Oracle Downgraded Legacy Encryption) attack [16]. Such misconfigurations of SSL can lead to vulnerabilities that can be exploited by attackers. The Qualys SSL Test rates server configurations on a range from A+ (the best) to F (the worst). An "A+" means that the server is configured securely against the most common SSL/TLS vulnerabilities and attacks; an "F" means that the server is vulnerable to the most common SSL/TLS attacks.

Our framework helps us identify the security and privacy issues of FL apps. It also helps determine how unsafe it is to deal with sensitive information in plaintext on either local storage (internal or external storage), or through network traffic, since both are prone to data leakage. This means that attackers can take advantage of the poor security design of FL apps to reveal and intercept much of user data.

5 ANALYSIS RESULTS

In this section, we show the results of our security and privacy analysis of 41 FL apps. At first, we start by examining all permission requested by FL apps. Second, we analyze all potential points for data leakages, such as network traffic and local storage analysis. Lastly, we test the back-end servers of corresponding FL apps for their authentication and authorization schemes.

5.1 Permissions Analysis Results

Table 2 shows the results of our permissions analysis of the FL apps. Also, it shows that all the FL apps requested access to the Internet, network, and location, none of which is surprising. FL apps rely on the Internet and location to function properly. This also means that many of them face the risk of leaking sensitive information over the Internet or insecurely storing location data on local storage.

Among the 41 apps, 85.4% asked for the "C2D_MESSAGE" permission to send and receive notification messages over the Google Cloud Messaging (GCM) server. The "C2D_MESSAGE" permission is used by some FL apps for communications. In addition, many FL apps use this permission to send invitation codes. Even though this permission is popular among FL apps, it does not raise security concerns for our study as the traffic to the GCM server is encrypted over HTTPS.

We found that many FL apps asked for permissions deemed dangerous. Android classifies permissions as dangerous if the permissions allow access to sensitive information that could potentially affect the privacy of the user or the security of the device [27]. For instance, 85.4% of the 41 apps asked to write to the external storage of the device, 58.5% of them requested to read the user's contacts,

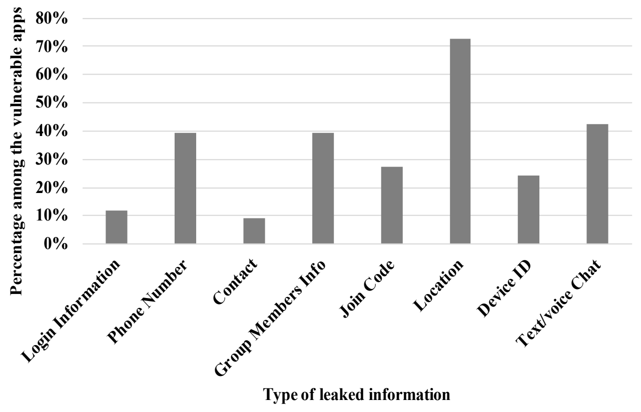

Figure 4: Percentage of 33 vulnerable apps and the types of leaked information.

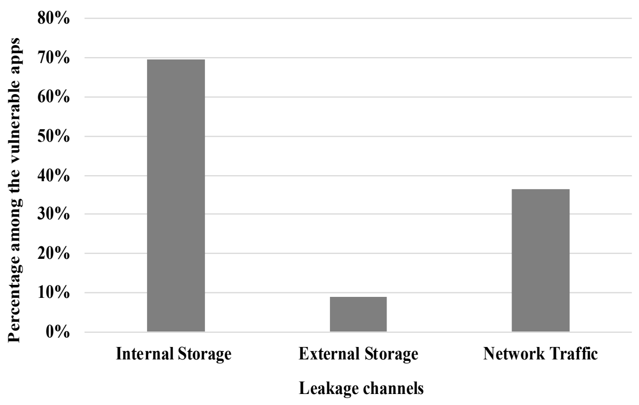

Figure 5: Percentage of 33 vulnerable apps related to sensitive information leakage channels.

53.7% of them asked to read the phone's status, and 51.2% of them requested access to the camera device. We highlight the dangerous permissions in bold in Table 2. Apps can use these permissions to access personal information or private resources, such as users' location and external storage. Furthermore, we found some of the FL apps asked for permissions that were deprecated. For example, the "READ_HISTORY_BOOKMARKS" permission was deprecated as of Android 6.0.

Due to these findings, we raise the question of how FL apps keep and transfer collected sensitive information of users. Maintaining the confidentiality of sensitive information is very important. FL apps with weak security mechanisms can leak private information.

5.2 Sensitive Information Leakage

Our study detected that 33 (80.4% out of 41) FL apps were vulnerable to leaking sensitive information in network traffic and/or local storage. Figure 4 shows the types of sensitive information that can be leaked from those 33 FL apps. Among them, location data was the most commonly leaked data, and it can be leaked by 72.7% of the 33 FL apps. An attacker can intercept or obtain user location data for a variety of malicious activities, such as determining when the whole family is away from home, or for other criminal purposes.

Out of the 33 vulnerable apps, 39.4% leaked phone numbers of users or group members. Some FL apps use the phone number to invite members to a circle over text messages. One of the apps used a free SMS API to send SMS messages that contained a join code for inviting group members. In such a scenario, an attacker can use the same API to invite people to a group while pretending to be a family member. The invited members would then receive an SMS message containing a join code from the same free API service. This can fool a family member into believing that the inviter is the same as a legitimate family member. Thus, a group member might unwittingly join an attacker's circle and share location and other data with the attacker.

On the other hand, around 12.1% of the 33 vulnerable apps leaked user credential information (e.g., username and password). With these apps, an attacker can intercept and steal the credential information to hack the users' accounts. After that, the attacker has full control of users' accounts for a wide range of malicious activities, such as obtaining credit card information, personal information, and family member information. There is another risk associated with the loss of usernames and passwords. This can be harmful, especially to people who reuse the same passwords. This kind of problem is very common [13].

Additionally, 27.3% of the 33 vulnerable apps leaked join codes in plaintext, either over HTTP connection (15.2%) or in local storage (12.1%). Any person who has the join code can join a family circle. This means if an FL app does not protect the join code, an attacker can intercept it to join a circle where a lot of sensitive information about users (e.g., location, chats, etc.) is readily available. Also, the attacker can interact with group members in order to harm them, such as setting up a meeting while pretending to be a family member for assault or rubbery. Therefore, leaking join codes in insecure forms can be hazardous to all group members (families, friends, and others).

We also determined that 42.4% of the 33 vulnerable apps leaked messages of group members in plaintext. This is a serious vulnerability since messages between group members might have sensitive information, such as credit card numbers, location data, security PINs for doors or lockers, and the like. Transferring or storing such conversations in an insecure channel can be harmful.

It was also noted that 39.4% of the 33 vulnerable apps leaked group member information, such as email addresses, phone numbers, full names, and locations. Other sensitive information, such as contacts and device ID (9.1% and 24.2%, respectively), was found unencrypted. These kinds of sensitive information can help attackers impersonate users or carry out phishing attacks.

5.3 Channels of Sensitive Information Leakage

We mainly inspected two points of data leakages: in transit or at rest. Figure 5 depicts the percentage of vulnerable apps and the corresponding leakage channels. Network traffic, internal storage, and external storage are the main channels. Internal storage is the major leakage channel, with 69.7% of the 33 vulnerable apps storing sensitive information insecurely. Intruders with physical access or root permission to the device can easily obtain such information if

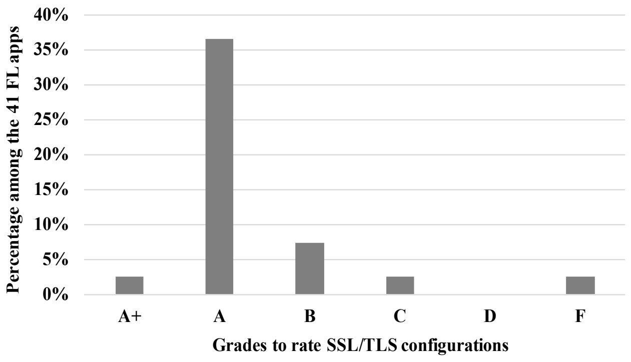

Figure 6: Results of the evaluation of back-end servers' SSL/TLS configurations. "A+" is the desired grade. "A" and "B" grades are acceptable. "C" is usually used for configurations that do not follow best practices. "D" and "F" grades are for servers with security issues.

data is not encrypted. Also, we found that 9.0% of apps stored sensitive information on the external storage, which is a shared storage space for all apps. One app stored group member information and another stored voice conversations between two group members. The risk of storing sensitive information on external storage is that any app that has the *read external storage* permission can read all the data stored there.

Furthermore, we found that 36.3% the 33 vulnerable apps leaked a great deal of sensitive information (e.g., login, join code, location, email, and phone number) at transit. A network attacker can intercept this information for malicious activities. This kind of interference can be relatively easier as the attacker does not have to create a malicious app or have physical access to the phone. The attacker can simply monitor the traffic using available sniffing tools, and perform harmful actions remotely. It is worth mentioning that some apps leak sensitive information on both local storage and over the network.

Overall, we found that most FL apps failed to protect users' data despite collecting a lot of sensitive data. This makes users vulnerable to physical and digital crimes, both of which endanger families. Most of the apps that we analyzed are used by millions of users who have a reasonable expectation of confidentiality and protection. It is relatively simple for developers of FL apps to incorporate security protection. However, our analysis shows that developers seem to prioritize app functionalities over security.

5.4 Leakage Analysis on Back-end Servers

FL apps need to communicate with back-end servers for information storage and retrieval. We found that 16 of the 41 FL apps communicated with back-end servers over HTTP which was insecure. Among these 16 apps, five had authentication or authorization vulnerabilities. For example, we were able to retrieve information about users and families from back-end servers by resending captured packets. For some of these 16 apps, we could retrieve PII using easily guessable information, as described in detail in Section 6.

Some of the FL apps that we analyzed depended on an access token to retrieve user data from the server. The access token is a sequence of numbers generated by the server when users log in with their credentials. Apps use this access token to authenticate the user to the server with every request instead of requiring usernames and passwords. The access token always has a designated lifetime. When the lifetime expires, the access token becomes invalid. Then, the server asks the user to log in again to generate a new valid access token. A short-lived access token is more secure than long-lived ones. However, we found that one server sets a lifetime for the access tokens that are overlong, such as seven days.

For FL apps that use SSL channels to communicate with back-end servers, we used the Qualys SSL scanner to detect server misconfigurations of SSL. Figure 6 shows the Qualys ratings for all HTTPS URLs found in the 41 FL apps. Most of the FL apps had back-end servers with a grade of "A" which is very good. Upon further analysis, we found that most of these apps used third party cloud services as back-ends servers, such as Amazon AWS, Microsoft Azure, and Google G-Suites. We found one app whose back-end grade was "F" which indicated that it was very vulnerable to Heartbleed and POODLE attacks. Overall, the common reliance of FL apps on popular third party cloud services makes it relatively easier to avoid misconfigurations. The question about whether there might be privacy violations by those cloud services is out of the scope of this paper.

6 CASE STUDY

FL apps face risks from two types of attacks, network and local storage attacks. In this section, we selected two of the nine apps that were found to leak join codes. As shown in the following case studies, one app leaked join codes over network traffic, and the other leaked join codes in local storage. These two apps have over 50,000+ installs from Google Play. For each app, we installed the app on two Android devices (*Version 8 which is known as Oreo*). Then, we created different user accounts on the two devices. Afterward, we initiated a circle on one device, and then we generated the join code to invite the other user. Later, we shared the join code with the second device via email. The join code was fixed for the circle, thus any person with the join code could join the circle. From the second device, we used the join code to join the circle. Finally, we started sharing location data and chats between the two devices.

Proof-of-concept of a network attack. The first app leaked the join code in an insecure form over the network. We intercepted all network traffic between the devices and the back-end server. Afterward, we analyzed the network traffic traces to find any hint of actual join codes. We found that the join code existed in the network traffic, as "code=xxxxxx". To demonstrate a network attack, we installed the same app on a third device for the purposes of launching an attack. We used the same join code that we had found in the network traffic. We succeeded in joining the circle by the third phone, and we were able to view the location of all the group members along with their chat messages.

Proof-of-concept of a local storage attack. The second app leaked join codes in an insecure form from the local storage. We extracted the entire database of the app from the device that generated the join code. We then browsed the app's database with a SQLite viewer on a host PC to view all the stored data [19]. We found that all the data was stored in plaintext, including the join code. Similar to what we did with our network attack, we used a third device for the attack. We installed the same app on the third device, and then we used the leaked join code to enter the circle. We succeeded in joining the circle and were able to view the locations of all the group members. It should be noted that attackers could also use an Android emulator, rather than a real device, to hide their actual current location when joining a circle. Using an emulator makes it hard for law enforcement to locate the source of attacks.

Authentication and authorization vulnerabilities in backend servers. We observed that some apps use short and easily guessable information in their HTTP requests to retrieve user data. For example, the app that we mentioned above that leaked join codes over HTTP sent the join code within the HTTP request to retrieve all user and group member information, including userID, password, location, and other PII. In fact, the app generated the join code using the same characters of the circle name. For example, if the circle name is "123", the app will generate the join code: "123". For this app, the HTTP request is similar to *http://xxxx/xxxx/getByMembers.php?userFamily=xxx*, and "userFamilly" is the same for the circle name and the generated join code. For the case study, we generated a circle and then invited group members to join the circle. Next, we used a Chrome browser on a host PC to send the same HTTP request to the server. We were able to reveal all information of every member of the entire group. The server did not require any kind of authentication, it did not even verify whether the request had come from a PC or a mobile device.

Additionally, this server did not check the strength of the userID value. As a result, attackers can perform dictionary attacks to access the circles that use short values as userID. It is worth noting that there were four other apps out of the 41 FL apps that had weak authentication and authorization mechanisms. Similarly, an attacker only needs to guess the userID to reveal and access sensitive information about group members. In all these five apps, the data returned from their corresponding servers was not encrypted.

7 DISCUSSION

7.1 Implications

Evaluating FL apps is critical. This will make an impact to the society, where many households nowadays rely on such FL apps to communicate and share data with their families. We think our findings are useful for both users and developers: for users to feel more secure and confident in the security and privacy practices of FL apps, and for developers to improve their products so they can build trust to enhance their financial profits and reputations.

The essential problem is that most FL apps do not use encryption mechanisms. There are three main parties each play a vital role in Android security: users, developers, and the platform (Android). Relying on only one party to provide the security and privacy measures may lead to defects in the others. For example, some users want to root their Android devices in order to have more freedom and control. Rooting devices may disable essential security mechanisms from devices which may open the door for adversaries to have high privileges and superuser access in devices. Consequently, adversaries could extract sensitive information if it was stored unencrypted on internal storage with minimum efforts. From another perspective, some developers focus on apps' functionality more than security and privacy. Some developers may not have experience on how to use security measures.

Another main concern about the current FL apps is privacy. Some parents may not be comfortable with using FL apps since a lot of the family's data is collected by FL apps. Indeed, there is a clear call for a trustworthy, robust, and secure FL ecosystem.

7.2 Suggestions

Providing effective suggestions to protect against data leakage in mobile apps can be a challenging task. From the users' perspective, user awareness is very important for their security and privacy. Users have to be aware of the maker of the mobile apps. For instance, users should seek answers to some questions before using FL apps: is the mobile app designed by a trusted entity, is it from trusted markets, what kind of data is the app collecting, and is the data collection relevant to app's features. Having answers to these questions can reduce the harm as users would raise or lower their expectations aligned with their findings. To elaborate, if an FL app is from a suspicious vendor and a user needs to use the app, a user can use some protections and give the minimum amount of privileges to the app.

Furthermore, a user should be aware of who joins a family's circle by identifying and verifying each member on the circle. Moreover, users have to distribute join codes amongst a group by using secure methods if FL apps are vulnerable to some of the channels discussed in this paper. For instance, a user can simply create a VPN channel in their phone to protect against the network vulnerabilities like what we found in our analysis. A VPN will create a secure channel to tunnel all the traffic inbound and outbound. This will encrypt all the traffic of the user's phone regardless of the app.

From the developers' perspective, app developers should implement best practices for security mechanisms. Protecting users' data is vital, especially if the data has personally identifiable information. To best protect users, data transferring and storing channels must be secured to prevent unauthorized access. For example, the mobile app should only transmit user data via secure channels, such as SSL connections. Furthermore, data at rest should always be protected by ensuring that data is stored securely on the device and on the server.

To mitigate join code leakages, FL apps should set a short lifetime for the join code as setting a long lifetime gives an attacker more time to crack the code. FL apps also should generate different join codes for each invitation to the circle, each of which can only be used once. This provides greater protection to the FL system. In contrast, generating a fixed join code creates multiple security vulnerabilities that let attackers compromise circles easily. Last, join codes should be generated randomly, e.g., using secure random number generators to avoid easily guessable join codes, as discussed in Section 6.

7.3 Limitations

The methodology used for this study allowed us to detect a number of security and privacy issues with FL apps; however, there were limitations in this research. The dataset contained 41 apps that were identified in the Google Play store. More candidate apps might have been located if we had utilized other app markets. However, the set of apps we used was representative as we chose popular FL apps with noticeable number of downloads. We plan to extend our crawler to download more FL apps from different markets for future work.

In our study, we analyzed the FL apps based on the free features available to all users. However, users can choose to purchase premium features, such as ones that allow for larger groups/circles, the storage of location history for more than one month, and others. Although we did not investigate the premium features due to budgetary restrictions, we think some security and privacy problems might exist with premium features as were identified with the free features. It is worth noting that app descriptions of the tested FL apps do not mention any additional security and privacy protections for the feature set for premium subscribers.

Another limitation involved the hands-on effort required to test the apps as the process needed a great deal of manual interactions, including: setting up an account, creating a circle, inviting group members, and generating/sharing join codes. Additionally, each FL app may have many join codes; thus, to locate join code leakage, it requires finding every join code generated be correlated with the FL app being examined. For future work, we plan to expand our work by automating our methodology to overcome the limitations caused by the need for manual interactions so that our methodology can be scalable.

To explore code branches, we used Android Exerciser Monkey to generate random UI events to interact with the 41 FL apps. Those code branches were not explored by the manual efforts mentioned above. However, some research has found that the Monkey tool is only capable of exploring about 60% of code branches [21, 22]. Thus, it is likely that the Monkey tool failed to explore some of the FL app code branches that may leak sensitive information, either over network traffic or in the local storage.

Android 10 was released in September of 2019. Scope access on external storage, or "scope storage" was first introduced in this release. An app that targets Android 10 and above can create a directory on external storage to store its private information, and other apps cannot access this directory or its data. However, based on [29], Android version 8 remains the most popular in use. However, some users may think updating devices to a new version might affect the performance, the familiarity, or certain useful functions available in their current OS. Therefore, some users may not update their Android phones.

7.4 Responsible Security Disclosures and Responses

Among the 33 FL apps with security vulnerabilities, five apps have been removed from the Google Play store before we contacted the developers. However, we have sent emails to the developers of the 33 vulnerable apps disclosing all the identified vulnerabilities in their apps. We did not write a group email for security disclosures, but wrote an email for each app's corresponding developer to maintain the confidentiality of our findings. In the email, we clearly stated the vulnerabilities and asked if there might be some interests by developers in addressing the security and privacy vulnerabilities. We received responses from 14 developers. Three of them were auto-reply responses (no human response). Two other responses came back with ticket numbers. When we inquired updates about the received tickets, we reached some web pages that did not have any information. We considered this negligence by developers an issue to the app development and integrity as security disclosures should always be expected.

The remaining nine responses we received were clearly written by developers as they asked for more details about the vulnerabilities and some suggestions on how to fix them. We then provided more details about the vulnerabilities in their apps. Nine of the developers replied and confirmed the security issues, and they showed interest in mitigating the issues in one of the next updates. One developer was very keen in addressing all disclosed vulnerabilities in a few weeks from our interaction.

Interestingly, one developer mentioned that their app relied on security mechanisms available by default on Android devices, and there is no need to have an extra level of security for users' data. To elaborate, the developer meant that even if data was stored unencrypted, attackers would not be able to reach the data unless the device was rooted or the user installed malicious apps. However, we argue that not implementing security measures to protect personal data and putting the burden on users to do so are not realistic as users might mistakenly download malicious apps and their data would be at risk [1, 15, 17].

8 CONCLUSION

Although there are many benefits associated with the use of FL apps, our research showed that there were serious security and privacy vulnerabilities in the majority of the apps we tested. In this work, we performed security and privacy analyses of 41 FL apps collected from Google Play to test how user data was collected, transferred, and stored. Consequently, we found many FL apps collected a lot of Personally Identifiable Information (PII). We evaluated the apps on their protection of confidential data during transfer and storage, and found that 80.4% of the apps stored PII (e.g., location data) and secrets (e.g., join codes) in plaintext. Attackers can exploit these vulnerabilities to obtain sensitive information from insecure storage or during insecure transfer. Additionally, around 15.1% of the apps leaked sensitive information from back-end servers as a result of the absence of authentication and authorization checking. An attacker can exploit these technical vulnerabilities on the back-end server to reveal users' sensitive information.

Clearly, these issues indicate that FL apps pose risks to the safety and security of the families who use them. Developers need to look into the design and implementation of these apps to address these issues and provide more secure experience to users. In future research, we plan to build tools to measure the security and privacy of FL apps automatically.

ACKNOWLEDGMENTS

We thank the reviewers for their valuable comments. The research effort of Khalid Alkhattabi and Ahmed Alshehri was supported by the Saudi Arabian Cultural Mission (SACM). The research effort of Chuan Yue was supported in part by the Army Research Office under Grant Number W911NF-17-1-0447.

REFERENCES

[1] Sherly Abraham and InduShobha Chengalur-Smith. 2010. An overview of social engineering malware: Trends, tactics, and implications. *Technology in Society* 32, 3 (2010), 183–196.

[2] 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.

[3] Amiangshu Bosu, Fang Liu, Danfeng Yao, and Gang Wang. 2017. Collusive data leak and more: Large-scale threat analysis of inter-app communications. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 71–85.

[4] Chiara Braghin, Stelvio Cimato, and Alessio Della Libera. 2018. Are mhealth apps secure? a case study. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. IEEE, 335–340.

[5] Rajchada Chanajitt, Wantanee Viriyasitavat, and Kim-Kwang Raymond Choo. 2018. Forensic analysis and security assessment of Android m-banking apps. *Australian Journal of Forensic Sciences* 50, 1 (2018), 3–19.

[6] Kim-Kwang Raymond Choo, Jody Farnden, and Ben Martini. 2015. Privacy Risks in Mobile Dating Apps. *Computing Research Repository (CoRR)* (2015).

[7] U.S. Federal Trade Commission. 2012. Children's Online Privacy Protection Rule. *Federal Register* (2012).

[8] Xingmin Cui, Jingxuan Wang, Lucas CK Hui, Zhongwei Xie, Tian Zeng, and Siu-Ming Yiu. 2015. Wechecker: efficient and precise detection of privilege escalation vulnerabilities in android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 1–12.

[9] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. 2014. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*. ACM, 475–488.

[10] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgrtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. ACM, 50–61.

[11] Dongjing He, Muhammad Naveed, Carl A Gunter, and Klara Nahrstedt. 2014. Security concerns in Android mHealth apps. In *AMIA Annual Symposium Proceedings*, Vol. 2014. American Medical Informatics Association, 645–654.

[12] Yangyu Hu, Haoyu Wang, Yajin Zhou, Yao Guo, Li Li, Bingxuan Luo, and Fangren Xu. 2019. Dating with scambots: Understanding the ecosystem of fraudulent dating applications. *IEEE Transactions on Dependable and Secure Computing* (2019).

[13] Kracker joshua. 2018. *ull-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL), protocols*. Retrieved February 1, 2020 from https://www.pandasecurity.com/mediacenter/security/password-reuse/

[14] Ansgar Kellner, Micha Horlboge, Konrad Rieck, and Christian Wressnegger. 2019. False Sense of Security: A Study on the Effectivity of Jailbreak Detection in Banking Apps. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 1–14.

[15] Jalaluddin Khan, Haider Abbas, and Jalal Al-Muhtadi. 2015. Survey on mobile user's data privacy threats and defense mechanisms. *Procedia Computer Science* 56 (2015), 376–383.

[16] Bodo Moller, Thai Duong, and Krzysztof Kotowicz. 2014. This POODLE bites: exploiting the SSL 3.0 fallback. *https://www.openssl.org/ bodo/ssl-poodle.pdf* (2014).

[17] Ildar Muslukhov, Yazan Boshmaf, Cynthia Kuo, Jonathan Lester, and Konstantin Beznosov. 2012. Understanding users' requirements for data protection in smartphones. In *2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE, 228–235.

[18] Achilleas Papageorgiou, Michael Strigkos, Eugenia Politou, Efthimios Alepis, Agusti Solanas, and Constantinos Patsakis. 2018. Security and privacy analysis of mobile health applications: the alarming state of practice. *IEEE Access* 6 (2018), 9390–9403.

[19] M Piacentini, P Morgan, J Miltner, R Ravanini, R Peinthor, M Kleusberg, J Clift, and JT Haller. 2015. DB browser for SQLite. https://sqlitebrowser.org/

[20] SSL Qualys. 2009-2013. *Labs: SSL Server Test*. Retrieved February 10, 2020 from https://www.ssllabs.com/ssldb/analyze.html

[21] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System. In *28th USENIX Security Symposium (USENIX Security 19)*. 603–620.

[22] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman. 2018. "Won't somebody think of the children?" examining COPPA compliance at scale. *Proceedings on Privacy Enhancing Technologies* 2018 (2018), 63–83.

[23] Erik Riedel, Mahesh Kallahalla, and Ram Swaminathan. 2002. A Framework for Evaluating Storage System Security.. In *FAST*, Vol. 2. 15–30.

[24] Rushank Shetty, George Grispos, and Kim-Kwang Raymond Choo. 2017. Are You Dating Danger? An Interdisciplinary Approach to Evaluating the (In)Security of Android Dating Apps. *IEEE Transactions on Sustainable Computing* (2017).

[25] Android Studio. 2017. *UI/Application Exerciser Monkey*. Retrieved February 10, 2020 from https://developer.android.com/studio/test/monkey.html

[26] tcpdump group. 2017. Android tcpdump [online]. (2017). https://www.androidtcpdump.com/

[27] Android team. 2016. *Permissions overview*. Retrieved February 10, 2020 from https://developer.android.com/guide/topics/permissions/overview#permission-groups

[28] Android team. 2016. *Save files on device storage*. Retrieved February 10, 2020 from https://developer.android.com/training/data-storage/files

[29] Android Team. 2019. *Distribution dashboard*. Retrieved February 10, 2020 from https://developer.android.com/about/dashboards

[30] Zack Whittaker. 2019. *A family tracking app was leaking real-time location data*. Retrieved February 10, 2020 from https://techcrunch.com/2019/03/23/family-tracking-location-leak/

[31] R Winsniewski. 2012. *Android–apktool: A tool for reverse engineering android apk files*. Retrieved February 10, 2020 from https://ibotpeaches.github.io/Apktool/

[32] Hao Zhang, Zhuolin Li, Hossain Shahriar, Dan Lo, Fan Wu, and Ying Qian. 2019. Protecting Data in Android External Data Storage. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 924–925.