Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

User Privacy & iPhones:

An analysis of user privacy on the iOS platform

Thomas Kelly
Supervised by Douglas Leith

April 23, 2021

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer Engineering)

Declaration

I, Thomas Kelly, hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write", located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

As smartphones and use of the internet become ubiquitous in modern society, the maintenance of user privacy in these contexts has become an increasingly difficult task. With Apple's iPhone accounting for approximately a third of smartphone users, a large portion of the world has become dependent on usage of iOS devices for staying connected in modern society. This has become particularly relevant in light of the Coronavirus pandemic. The success of iPhones is largely founded on the wide range of applications available for download that grant the device additional capabilities. From popular social media apps like *TikTok*, to the contact-tracing apps created in many parts of the world to combat the spread of COVID-19, applications have become an important part of daily life worldwide. It is therefore of upmost importance that users be made aware of what personal data these apps collect about them, and for what purposes they are collecting it.

This work firstly focuses on an analysis of how user data is handled by Apple on their popular iPhone device. Subsequently, the same investigative techniques are applied to popular iOS applications allowing them to be examined from a perspective of privacy. While many applications are quite respectful and secure in their handling of sensitive user data, this work finds areas in which security & privacy measures are in some cases lacking, and highlights areas meriting concern. The apps considered in this analysis are *COVID Tracker Ireland*, *Aarogya Setu*, *Airbnb* and *TikTok*. The findings of this work largely indicate that users should be more concerned about their privacy, and in several cases are left uninformed as to the length of network activity carried out on their iOS device. Apple, alongside the applications discussed in this work, all have areas in which their security measures could be improved, assuring the privacy of their users is maintained to the highest available standards.

Acknowledgements

I firstly want to extend my gratitude to the supervisor of this work Douglas Leith, for his continued guidance and insight throughout the course of this dissertation.

I also cannot thank my close family and friends enough, who have been unwavering in their encouragement and support during my five years at Trinity.

Contents

1 Introduction **8**

2 Background **10**

2.1 An overview of user privacy 10
2.2 Related work 11
2.3 Sensitive Data 14
2.4 Threats to user privacy 14
2.5 Security Overview 15

3 Experimental Setup **18**

3.1 Device Configuration 18
3.2 Jailbreaking 19
3.3 Cydia Substrate 19
3.4 SSL Killswitch 2 20
3.5 Frida 21
3.6 mitmproxy 24
3.7 Hopper Disassembler 26

4 Evaluation **27**

4.1 Apple 27
a. Apple Identifiers & Sensitive Information 28
b. iCloud 28
c. Advertising services 31
d. App Store (iTunes) 35
e. Find my iPhone (FMIP) 36

f. Idle connections . . . 37
g. Maximising Privacy . . . 39
h. Observations . . . 41
4.2 COVID Tracker Ireland . . . 42
a. Interactions with the app . . . 42
b. Registration . . . 43
c. General Usage . . . 45
d. Observations . . . 49
4.3 Aarogya Setu . . . 49
a. Interactions . . . 50
b. Google Firebase connections . . . 51
c. Aarogya Setu connections . . . 54
d. Observations . . . 60
4.4 Airbnb . . . 62
a. Registration . . . 62
b. Third Parties . . . 64
c. General Usage . . . 70
d. Observations . . . 73
4.5 TikTok . . . 74
a. Gaining entry . . . 74
b. Registration . . . 75
c. General usage . . . 80
d. Third Parties . . . 87
e. Observations . . . 88

5 Conclusion **91**
5.1 Future Work . . . 91
5.2 Conclusion . . . 91

Bibliography **93**

Appendices **101**

List of Figures

2.1 Overview of SSL operation, created with *Lucidcharts*. 16

3.1 Depiction of Frida's interception strategy (25) 22
3.2 Diagram of how Frida stalking works, wrapping basic operation blocks. (25) 23
3.3 Intercepted traffic as displayed in *mitmproxy* CLI 24
3.4 Steps of MITM attack, made with *Lucidcharts* design tool. 25
3.5 Hopper: Disassembling *TikTok* and searching for an 'encrypt' methods . 26

4.1 Body of POST request to /c2, displaying activity reporting. 30
4.2 Apple Advertising settings and information 33
4.3 COVID Tracker App display of infection statistics. 47
4.4 Aarogya Setu display of infection statistics. 50
4.5 *Airbnb's* privacy dialog upon registration for the app 63
4.6 *TikTok's* For You page (FYP). 82
4.7 Interception of class-method *HMDHeimdallrConfig performanceReportURL* 86

List of Tables

3.1 Summary of device settings 19

4.1 Summary of device settings 40

CHAPTER 1

Introduction

With an increasing portion of human interaction and communication taking place online in the modern world, the privacy of internet users has emerged as a pivotal issue for the evolution of the web and the technology that relies on it. With 728 million people in the world using iPhones (10), Apple controls about a third of smartphone user's data. With a reputation of being secure and trustworthy, Apple, and more specifically iPhones have prompted only a small amount of speculation and research around how they handle their user's privacy. The success of smartphones has also largely been driven by the myriad of popular applications supported by them, making them a crucial part of how our society functions. The ever-increasing popularity of these devices & applications has prompted concern as to how dependent we are on this technology, and how much we should trust it with our personal information.

In light of the COVID-19 pandemic, this concern has been accentuated in many cases through development of COVID tracker apps. Taking the example of India, the government made downloading their COVID app *Aarogya Setu* mandatory in May 2020 (9), unlike any other democracy in the world. There has been much speculation and concern directed towards its handling of user data in terms of storage, but also in how it monitors users and shares their information with the Indian government. Many countries adopted these COVID tracker apps as a way to control the spread of the virus, prompting varying degrees of privacy concern depending on the app. While COVID apps will not be in use forever, popular social media apps like *TikTok* have become pervasive in modern society and have drawn a lot of attention regarding the security and privacy of their users. So much so that former US president Donald Trump

threatened to ban the application over concerns about user privacy and the app's Chinese origins (58).

With such a large portion of the world population utilising and relying on these applications for many different reasons, little work has been done formally analysing how these applications and the iOS devices they run on handle their user's data. This project examines the data transmitted in an iPhone's normal operation to Apple servers, while also analysing how four different currently topical and popular iOS applications handle user privacy. To perform this analysis, white-hat hacking techniques are employed, using a man-in-the-middle attack to intercept internet traffic spawning from an iPhone and analyse it from a perspective of privacy.

CHAPTER 2

Background

As the influence and power of the internet increases, attention must be drawn to the vulnerabilities we expose through expanding our online presence. As a core focus of this work, it is important to firstly discuss the state of user privacy in the world today, how it is protected, and its threats. As this project focuses specifically on iPhones, the following discourse largely regards privacy as it relates to Apple and iPhones.

2.1 An overview of user privacy

As one of the most influential laws protecting the privacy of internet users, it would not be proper to proceed without brief mention of the European Union's 2018 General Data Protection Regulation (GDPR) (42), which is the law enforcing the compliance of data collectors with EU standards of user privacy. Among the main definitions of the regulation are the rights people possess regarding their personal information, outlining that users must provide 'freely given, specific, informed and unambiguous' consent to data controllers to allow them to process data. While this and other measures enforces companies to be transparent and honest about their actions in privacy policies, it does not entirely prohibit user tracking or any other less desirable communications of user data, once it is outlined in the authorities' privacy policy.

While threats of hacking are not completely impossible, the majority of companies handle and store their user's information securely, encrypting sensitive material and using storage on protected servers. While security mitigations and compliance with best practices reduce the risk of an attack, hacks such as SQL injections and DDoS still exist

as threats. In this manner, immense trust is placed in tech companies to protect their user's identities and sensitive information. It is therefore pertinent that app vendors and servers collect the absolute minimum amount of data necessary for providing the service in question.

Placing laws and best security practices aside, undesirable transmissions of user data is still hugely relevant, particularly when it comes to the subject of advertising. As will be discussed in the following section, some iOS and Android applications have been shown to transmit sensitive user information to third parties. Examples of such practice are enabled by services like Cuebiq (41), who are a company that specialise in offering a service that allows app developers to track users in different ways, for the purposes of improving and understanding advertising strategies and their impact.

Having briefly introduced the current state of user privacy in today's world, the following sections discuss existing work related to the analysis and examination of iPhones and their applications, and provide further context and detail on specific privacy concerns and avenues by which user privacy can be exploited.

2.2 Related work

While Apple boast a strong commitment to maintaining user privacy and upholding high levels of security, iOS like any platform has a history of security vulnerabilities that have been exploited resulting in user data being stolen. One of the more popular works regarding user privacy as it relates to iPhones is 'iPhone Privacy' (43). In this work, N. Seriot (2010) documents how malicious apps downloaded through the App Store can access a significant quantity of sensitive data, and also includes discussion of the history of iPhone privacy concerns - most of which have since been addressed by Apple. Seriot also discusses the importance and sensitivity of device identifiers like the iOS Unique Device Identifier (UDID), and International Mobile Equipment Identifier (IMEI), both of which can be used to uniquely identify an iPhone, and should be highly protected. Tools have been developed such as PiOS, as described by Egele (44), with the aim of detecting privacy leaks of this sensitive information within iOS applications. However it is generally quite difficult to detect apps leaking this kind of data, and indeed some apps require this data and ask permission to access it, as they should. In the event user data is leaked, third party entities can use the data for a host of undesirable purposes, unbeknownst to the user.

For the most part, apps taking and storing this data are ones widely regarded to be completely legitimate and trustworthy. A majority of popular apps implement security in strict fashion and are highly respectful of user privacy, taking measures to obfuscate

and anonymise user data. However in the case of obfuscating data like user location, M. Srivatsa & M. Hicks (30) demonstrate that efforts like these can be effectively defeated by using location traces and a social network graph, determining the user's location based on who they meet. Another study conducted by Smith (5) observed that of a variety of popular apps considered, 68% transmit an iPhone's UDID to a backend server. Smith points out the many undesirable ways in which such data can be used once acquired (targeted advertising, user tracking). In *'Privacy in Mobile Apps'* (39) A. Pultier examines 21 popular applications in similar fashion to this work however on an Android device. A core discovery of this work is that 'mobile apps not owned by big American tech companies, (e.g. Google, Facebook) - apps such as VG, sports apps and dating apps - often communicate and share personal information with many third-party trackers' [39, p. 9].

App vendors can theoretically use UDIDs along with other user information to easily match devices with the phone owner's real identity, which would allow the basis for relatively simple real-time user tracking. Smith postulates that it would be quite trivial to implement a system that can physically track iPhone users, with a combination of UDIDs and timestamped IP addresses, since iPhone users still have no way of blocking the visibility of their UDID. There is however no evidence that this occurs, only that theoretically it is possible. This particular study highlights how usage of this information can be used for advertising tracking - however the process can be somewhat limited in iPhone settings as will be shown later, but not completely stopped. Another insight from Smith's work highlights that in the case of one the considered apps (ABC News app), a long-lasting cookie that persists for 20 years is given to each user, which could technically be used to link new UDIDs from old, even when a user switches phones. Practices like this make it difficult for users to escape being tracked by app vendors. Similar findings for the relevant apps are presented in this work.

In light of the quick roll-out of COVID Contact Tracing apps, much work has been done evaluating individual apps, alongside the Exposure Notification System (ENS) (69) produced through collaboration of Google and Apple. Among the first COVID apps to be produced, the Singaporean (35) and Australian (45) COVID apps drew initial concerns and attention from a variety of academic work (Michael & Abbas (46), Cho et al. (47)). Through ENS, app developers in these cases are enabled to use a partly decentralized contact tracing system operating over Bluetooth. The primary goal with such contact tracing apps is achieving a balance between high effectiveness and high levels of user privacy. Cho discusses the case of Singapore's *TraceTogether*, which requires citizens to place trust in the government in the absence of a fully decentralized peer-to-peer system, like many contact tracing apps around the world. However, it does not collect more data than is necessary for good functionality, deeming Bluetooth

information sufficient for finding close contacts. This is contrary to that of India's *Aarogya Setu*, which is found to monitor user's exact time-stamped GPS location, which will be discussed in detail further into this report.

Examining a variety of COVID apps deployed across the world, Sharma & Bashir (48) state that out of 50 considered COVID apps, 30 require permission to access different types of user data. This ranges from phone contacts, to photos, to device IDs and even image galleries. Another finding is that only 16 out of 50 apps are found to take security measures to encrypt and anonymise their user's data. What's more, Dehaye & Reardon (51) present evidence that an SDK utilised by many contact-tracing apps introduces security vulnerabilities by way of Bluetooth - even if the user has opted out of the contact tracing system.

Leith & Farrell (49) involves similar analysis to that of this work on the Android platform, concerning a variety of COVID apps. Network traffic for the Irish app *COVID Tracker* is clearly outlined, providing the grounds for comparison with the Android version of the app. Attention is drawn to the fact that requests made by the app can be linked together through reuse of tokens, which from a privacy point of view is not optimal. The app however handles user data suitably, as found in the Data Protection Impact Assessment (DPIA) (50) for the *COVID Tracker* app, which outlines all of the user data taken and monitored by the app (50 p. 11). This includes specifications of the frequency at which data is taken, retention and where it is stored. More vague representations of personal data are used to collect stats, two examples being age ranges in place of exact date of birth, and the use of a general locality to represent where a person is located instead of GPS coordinates.

All of the aforementioned works designate areas of privacy concern around iPhones and popular contemporary applications, upon which this work can use to identify specific areas of interest. While privacy concerns have been raised around the transmission of sensitive data from iPhones, there is currently no work specifically examining what internet traffic is being generated between iPhone devices and Apple servers, and what user data is being transmitted. Furthermore, while analysis of many COVID apps has been conducted on the Android platform (46), (47), (48), 52, no in-depth analysis of the user privacy of COVID apps has been conducted on iOS devices. The *COVID Tracker* app is taken and analyzed in order to compare it with that of the android analysis as observed in (49), while *Aarogya Setu* provides completely unchartered territory in terms of privacy analysis at the time of writing this dissertation. Similarly, the application of such techniques to popular contemporary applications like *Airbnb* and *TikTok* have not yet been implemented.

2.3 Sensitive Data

There are many types of sensitive data associated with an iOS device. iOS devices of course store lots of personal information about their users like name, address and age, but also information about how the user interacts with their device. This might include usage patterns and behaviours. Other types of sensitive data include device data, such as the aforementioned UDIDs and IMEI, and also hardware serial number - which can all be used to uniquely identify a particular iPhone. Each Apple account also has a Digital Services Identifier (DSID) associated with it, which uniquely identifies an Apple user's account which they may access across multiple iDevices. Many related works (4), (30) - as discussed in the previous section section - have shown that the acquisition of device identifiers such as the UDID can be used for malicious purposes, such as real-time user tracking. Other undesirable activities might include advertising tracking, whereby app vendors collect user data to be used for the purpose of targeted advertising.

2.4 Threats to user privacy

Lots of processes on an iPhone communicate user data. Of course, there are lots of processes that require this data and are transparent about its communication. Most apps require information about a device such as its model, screen size and software version to ensure it operates correctly. Some apps even require location to function as intended, and are fully transparent about this, such as a GPS or Maps application. Clearly, these are not infringements of user privacy.

There are however many reasons to be concerned about the collection of user data. A core issue is that of the case where data can be linked with a user's real-world identity, which prompts a variety of concerns. Smith's work (5) has established that it would be trivial to implement a system that could physically track iPhone users, with a combination of UDIDs and time-stamped IP addresses. Tracking can however occur when any user data, such as address, name etc. is transmitted alongside a token or app instance identifier of some sort. When subsequent connections occur within the app, the identifier or token is used as authorisation, and thus all of the connections can be linked together by this single token. In this manner, the user's anonymity is made vulnerable. The user need not even enter any details, since their IP address is included in every connection their device makes. Since IP address is an indication of the user's location, this can be used to determine the user's location over time when combined with other data. Furthermore, efforts made by app vendors to anonymize user location data have been shown to be effectively defeated as previously discussed (30). With all of this in mind, an attempt by a third party service or entity other than Apple to collect sensitive

information like UDID, Serial number or DSID should draw attention and should be appropriately explained.

Other cases which may prompt concern occur where app vendors collect sensitive information which is unnecessary altogether, or excessively accurate. For example, some applications may require the user's general location, perhaps defined by the user's country to function correctly. As an example, which is discussed later in this work, the Indian COVID tracker app *Aarogya Setu* collects exact GPS coordinates of the user, when a more obfuscated location reading (or none at all) would be sufficient for the app's intentions.

Of course, hackers and other bad actors pose a constant threat. By using an iPhone and iOS internet applications a user is agreeing in a lot of cases to the storage of their data on certain companies servers. Because modern applications themselves rely on products from other companies, user data can often be transmitted to third party companies, as will be shown at length in the app analysis sections below. In this manner, once personal information is on the internet and stored on servers, it is vulnerable to attack. Of course the more places a user's data is stored, the more vulnerable it is to such an attack. Events such as the occurrence of a large Facebook account leak in 2019 (68) serve as reminders that companies still cannot absolutely guarantee the security of their user's data, although they will claim to. It is therefore a focus of this work to discover exactly what user data is being sent to back-end services for the different apps in focus, and whether or not the privacy of users is preserved in their internet transactions using an iPhone.

2.5 Security Overview

Departing from the topic of privacy, this section discusses the main security measures used by an iPhone while communicating on the internet, and briefly introduces how these measures are circumvented in order to perform analysis. While a variety of security protocols and techniques exist for securing app user's data, the most relevant and important protocol for securing application network traffic is Secure Sockets Layer (SSL) technology. SSL is a technology that secures connections to websites over the *https* protocol. Internet connections are secured using SSL certificates, which are small files that cryptographically establish an encrypted connection between a web server and a connecting client entity. Servers receive SSL certificates from a Certificate Authority (CA), which is a company that validates the legitimacy of the entity that owns the server and issues them with a signed certificate. When a client application connects to a server, an SSL handshake is performed, whereby the server sends its certificate, which the client - in this case an iPhone - can validate against a list of trusted CAs which

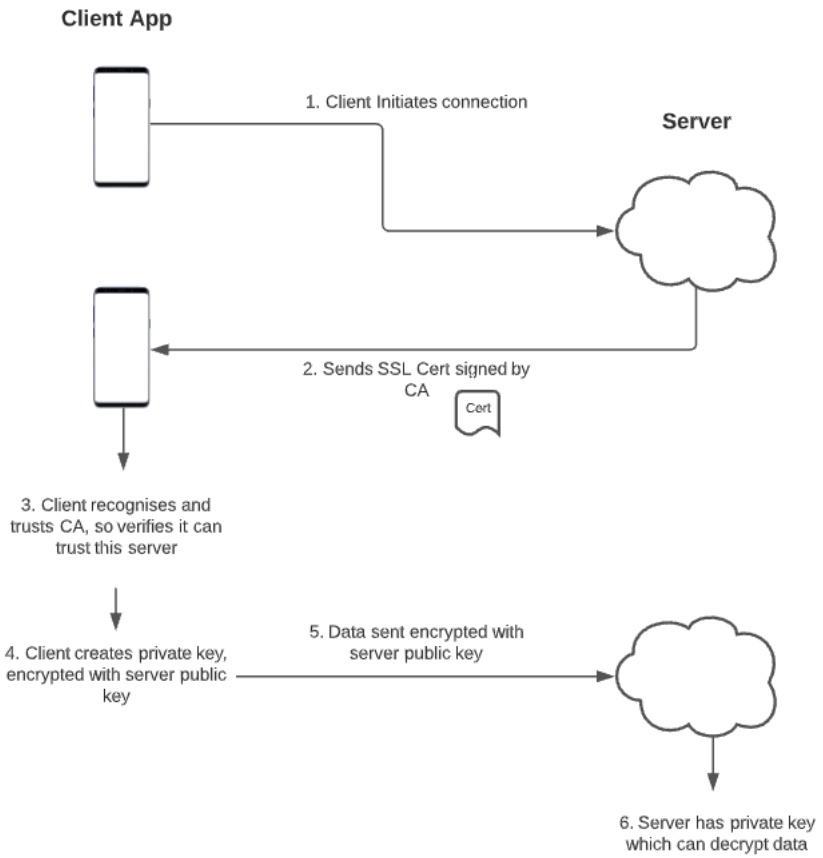


Figure 2.1: Overview of SSL operation, created with *Lucidcharts*.

Apple has pre-installed on the device (1).

After the SSL handshake process - which is depicted in figure 2.1 - the client has verified that it can trust the server, and generates a private key for itself which is encrypted with the server's public key. Now, both the client and server can encrypt information with the public key, which is only possible to decrypt with the corresponding private keys. A secure connection has been established.

In order to provide an extra layer of security against a hack such as a man-in-the-middle (MITM) attack, applications and other client-side processes can employ certificate pinning. This is implemented on the client-side, whereby the server's public key and other relevant information is pulled from the certificate into the client-side code. When future connections occur to the server in question, the client will validate that the incoming certificate matches the one it has 'pinned'. In the case it is not, the client will terminate the session.

To be able to perform privacy analysis on the traffic spawning from an iPhone, these measures need to be circumvented. The challenges posed by these security mechanisms are therefore two-fold.

- Intercept and log internet traffic, bypassing SSL security
- Unpin (disable) any certificate pinning that occurs

A man-in-the-middle or MITM attack is suitable for overcoming these mechanisms. A MITM is an entity that sits between the client and server during an exchange, and enables the interception of all internet traffic. In order to address the issue of certificate pinning, tools such as *SSL Killswitch 2* (2) and *Frida* (8) can be used to nullify the effect of pinning procedures. The MITM attack and other necessary measures are discussed in greater detail in Chapter 3.

CHAPTER 3

Experimental Setup

3.1 Device Configuration

The experiments performed in this work were conducted on an iPhone 8 running iOS 14.1, jailbroken with *checkra1n* (6), which is an iOS jailbreaking tool. Without a sim card, the device was not connected to a mobile network, instead using WiFi to access the internet. The majority of optional iPhone settings were left untouched in their default state, however an Apple account was signed-in so its traffic could be observed, and so apps could be downloaded from the App Store. This would also allow the phone to interact with other Apple services like *iCloud* and the *Find my iPhone* service (52).

Location services were enabled, alongside the *'Personalised Ads'* setting, which enables the capability for delivering relevant ads to the user, based on their behaviour and profile. This was done in order to allow observation of traffic spawning as a result of such settings. Sharing of analytical data was disabled, as was the default option for the iPhone being used. Another relevant setting was that of the *'Allow Apps to Request to Track'* option - which if enabled allows apps to ask permission to track the user across apps and websites owned by other companies. By default, this was enabled.

Setting	Configuration
Network Connection	WiFi
Location	Enabled
Cross-app Tracking	Enabled
Analytics	Disabled
Apple Advertising	Enabled
iCloud	Signed-in
Sim Card	None

Table 3.1: Summary of device settings

3.2 Jailbreaking

Jailbreaking an iOS device involves an escalation of user privileges on the device. It is necessary in order to install the required tools for the MITM attacks to succeed, alongside enabling the operation of tools like *Frida* and *Cydia Substrate* (7). The purpose of jailbreaking an iOS device is to take control of the root and media partitions. This is where all important system files are stored, and from where the jailbreak software can have maximum capability. An example of the capability offered by a jailbreak is showcased by that of the operation of *Cydia Substrate*. This is a runtime manipulation tool, allowing developers to build tweaks and patches that can modify processes on the iPhone and enhance the device's capability. The following section discusses *Cydia Substrate* in more detail.

3.3 Cydia Substrate

Cydia Substrate is the code modification platform behind *Cydia* (29), and is similar in a lot of ways to *Frida*, which will be discussed in more detail in a later section. It provides the foundations for other tools like *SSL Killswitch 2* to operate. Both tools rely on the escalation of privilege achieved in jailbreaking the target device, however both use a different approach to achieve their goals. The operation of *Cydia Substrate* is centered around three major components.

MobileHooker

This is a component used to replace system functions, and is known as hooking. A built-in device function called *method_ setImplementation* is used to enable the dynamic replacement of the implementation of an Objective-C method, with iOS code. The API that carries this out is called MSHookMessageEx(). A class can be hooked by passing it

to MSHookMessageEx() and designating a replacement class, which can be modified. As a result, system method implementations can be replaced with custom functionality.

MobileLoader

MobileLoader enables the loading of third-party code into a running application. While *Frida* injects by hijacking a thread of execution in a process, MobileLoader works by loading itself into the application using an environment variable *DYLD_INSERT_LIBRARIES*, which will result in it being called when the application is running.

Safe model

Safe mode provides a way in which to catch any third party apps that introduce crashes or bugs. This is handled by the MobileLoader, which will catch any extensions that crash the SpringBoard and put the device in safe mode. The SpringBoard is simply the application that manages an iPhone's home screen.

These components set the foundations for tweaks to be created for modifying an iPhone's runtime operation. An relevant tweak crucial to the success of this project, is that of *SSL Killswitch 2*.

3.4 SSL Killswitch 2

SSL Killswitch 2 is a tool used to disable SSL certificate validation on iOS. It is a software package that is transferred to the iOS device and accessed through an *ssh* connection to the iPhone. It patches low level SSL functions to override and disable the iPhone's default certificate validation routines. This ultimately bypasses certificate pinning implemented by apps, allowing the basis for a MITM entity to view the contents of internet traffic. The fundamental operation of *SSL Killswitch 2* is enabled by *Cydia Substrate* hooks.

The Secure Transport API is the lowest level implementation of TLS (SSL's successor) on iOS, handling certificate validation. The killswitch works by hooking and modifying three functions within this API, as described below.

- SSLCreateContext() → disable the built-in certificate validation in all SSL contexts
- SSLSetSessionOption() → remove the ability to re-enable the built in certificate validation

- SSLHandshake() → Force a trust-all custom validation
  - Works by making sure certificate checking/pinning doesn't get triggered

Once installed, the killswitch can be simply enabled in the iPhone's settings. *SSL Killswitch 2* does not however cater to applications that implement custom SSL pinning, which draw upon security libraries that *SSL Killswitch 2* doesn't account for. This is but one example of the many reasons for why *Frida* is required for further reverse engineering.

3.5 Frida

Frida is a command line tool used by developers, security researchers and reverse engineers. *Frida* allows the injection of JavaScript code into processes running on the iPhone, alongside other tools. This enables a wide variety of functionality, allowing the user to gain more knowledge of how the processes running on the phone operate. Apps on iOS are stored in *ipa* files (iOS App Store Package). When an app process is started, the IPA file is decrypted and its image is loaded into memory. *Frida* works by hooking function calls in this memory image. This can enable the user to 'hook' and tweak application processes, which users can use to discover more about a process, or modify its operation.

Frida's core of instrumentation is called Gum (23). It is written in C, and controls APIs that allow the functionality offered by *Frida*. When the target device is a jailbroken iOS device Gum can interact directly with otherwise restricted system commands. Alongside Gum, *Frida* makes use of some of its other libraries to package JavaScript code and inject it into existing software on the iOS device. In order for this to work, the software package *frida-server* must be running on the target device, which is a daemon that exposes *Frida's* capabilities over a TCP connection. As described in *Frida's* documentation (24) this process sets up the capability for a 'two-way communication channel', which allows the injection of custom JavaScript code, while also enabling the communication of information about the device and its running processes. The inner workings of the *Frida's* operations can be broken down further around three core principle, as described by a presentation from its developers (25).

Injection

During the injection process, the linux command *ptrace* is used to attach to a thread in a remote process on the device. This process thread has now been hijacked. Memory is then allocated for a bootstrapper function, which makes use of the *mmap* (27) command. Custom code is then loaded into the bootstrapper using *dlopen* (28). At this

point, the bootstrapper is executed, which starts a fresh thread in the remote process. This thread sets up FIFO communication to a debugger process, from which the injection operation can be monitored. Subsequently, the thread loads an '.so' file containing the *Frida* agent. The agent is executed before the FIFO pipe is closed, and the main thread execution is resumed. At this point, the custom code has been injected and is controlled by a file called 'frida-agent.so'.

Interception

The second angle of attack employed by *Frida* is that of an interception strategy. The notion of a 'caller' and 'callee' is integral here, illustrated in a diagram from (25) (p. 19).

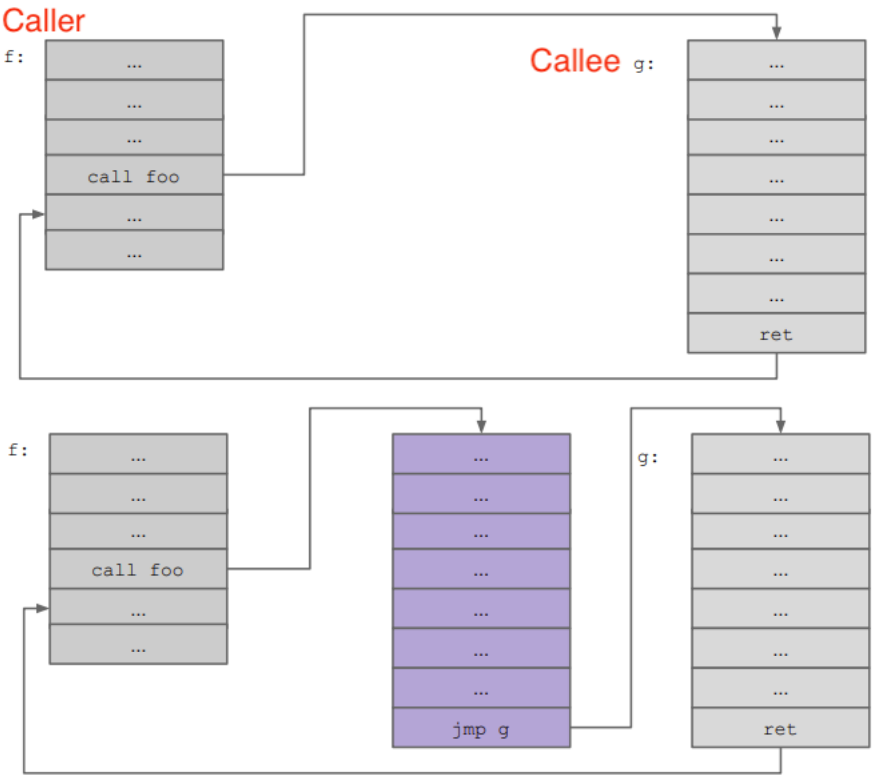


Figure 3.1: Depiction of Frida's interception strategy (25)

The address of the function to be intercepted is firstly determined. A middle-man of sorts is used, called a 'trampoline' which is introduced as an intermediate between the caller and callee. The trampoline calls the interceptor function, and hides all stack and register modifications. In this fashion, the interceptor function can be used to modify or monitor the targeted function. This allows for iPhone processes to be intercepted for analysis or modification.

Stalking

It is sometimes desirable to stalk or continually monitor a function's operation. The *frida-trace* (26) command is an example of this, whereby certain classes or methods can be targeted and closely monitored for when they are called and what parameters they pass. Looking deeper into the process of stalking a function, the list of operations performed in the function are taken and split into blocks. Each block of functionality is then wrapped in instrumentation which calls a stalker process for every basic block transition. This allows the targeted functions to be closely monitored.

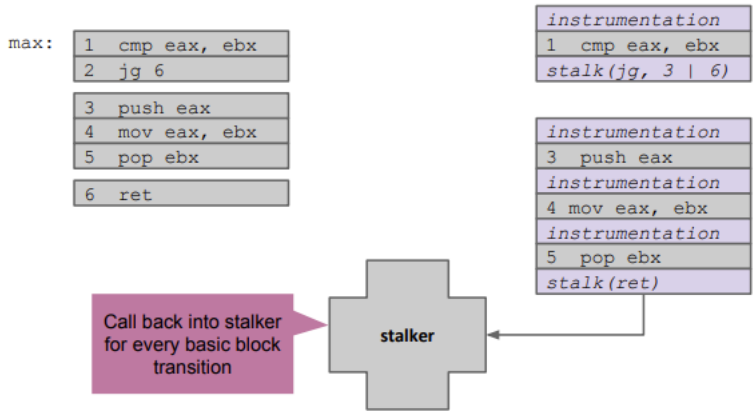


Figure 3.2: Diagram of how Frida stalking works, wrapping basic operation blocks. (25)

These components are all tied to together and made available to user's through *Frida's* CLI interface. This allows users to run *Frida* commands to interact with a device connected over USB. Some of the more commonly used commands are described briefly below.

Frida Commands

`frida-ps -Ua`

This command enables discovery of running processes on the iPhone (connected over USB), and their process identifiers (PIDs).

`frida-trace -U COVID Tracker Ireland -m "-[*class* *method*]"`

Executing this command will monitor method calls within the COVID app process, spawning from the entered *class*. Strings can be entered here, which enables the tracing of functions that contain this string as a substring. For example, entering the *Send* string in the *method* position above would trace calls related to the *NSMetricsSend* and *SendCOVIDStats* functions. Subsequently, *Frida* will display these calls whenever an action is performed on the iPhone that results in one of them being

called. This is an important strategy for attempting to find out more about how an application works.

```
frida -U cloudpaird -l {scriptToInject}.js
```

As has been previously referenced, scripts can be written to modify the implementation of certain functions and injected into processes. The above command injects a script into the *cloudpaird* process on the phone. A script can have many purposes here, for example unpinning custom certificate pinning, or to analyse the execution of a function in more detail. A more frequent application of *Frida* for this work was the latter reason above, whereby a script such as that of *observeMethod.js* (see **see Appendix B)** is injected to examine what parameters are passed to certain functions, and also see what values are returned by those functions. This enables the user to attain a much better understanding of what data applications handle.

3.6 mitmproxy

mitmproxy (3) is the software tool used to perform the MITM attack. *mitmproxy* is set up in transparent mode (4), whereby the the man in the middle, or device intercepting the iPhone traffic, is configured as a WiFi access point (AP). The iPhone connects to the AP running *mitmproxy* and configures it as its default network gateway. The iPhone device therefore isn't aware of the transparent proxy, however all of its traffic is configured to be routed through the AP. A certificate for *mitmproxy* must also be installed on the iPhone and listed as a trusted certificate. After successfully installing *mitmproxy*, the simple command *mitmproxy* can be run and the AP device can intercept and log all of the target device's network traffic. Figure 3.4 and following discussion details the underlying steps involved in a successful MITM attack.

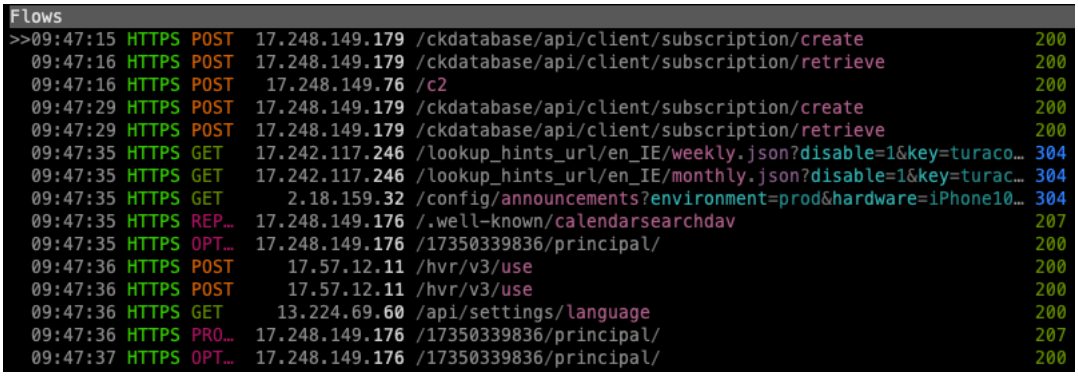


Figure 3.3: Intercepted traffic as displayed in *mitmproxy* CLI

As was discussed in section 2.5, after the client initiates a connection with a server, the server will respond with an SSL certificate, signed by a Certificate Authority (CA). This contains the public key necessary for encrypting any information shared between the

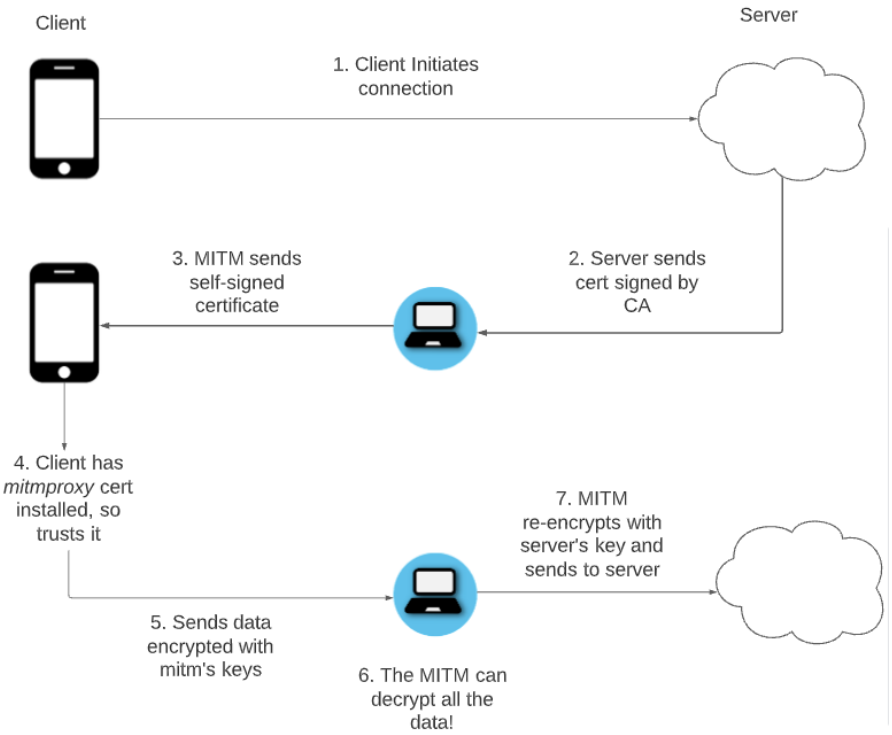


Figure 3.4: Steps of MITM attack, made with *Lucidcharts* design tool.

two entities. The MITM intercepts this, and instead sends its own self-signed certificate to the client. Provided the client device has the *mitmproxy* certificate installed as a trusted certificate, the device will accept this as the server certificate unless it fails any further attempts to secure the connection, such as SSL pinning checks. It will proceed to encrypt network data with the MITM's public key, which can be decrypted by the MITM, since it knows the corresponding private key. The MITM can now inspect the client's network traffic, before subsequently re-encrypting the data with the server's public key (which it intercepted earlier) and sends this to the server. Both the client and server parties are now under the impression they are communicating directly with one another, while the MITM intercepts all interactions between them.

With *SSL Killswitch 2* enabled on the device and *mitmproxy* running, network connections spawning from the iPhone can be logged, enabling examination of the contents of their requests and responses. While *SSL Killswitch 2* unpins most of the connections, some connections require more work to unpin, as has been mentioned previously. The solution for this is, as has been stated before, is the injection of scripts using *Frida*.

3.7 Hopper Disassembler

Hopper Disassembler (*Hopper*) is a tool which allows the disassembly of a iOS app binary file into assembly code. In order to get an app's binary, a *Frida* script such as that created by *AloneMonkey* (53) can be injected which extracts the file and dumps it on the injecting device. After that, the app can be processed by *Hopper*, which presents the app's assembly code.

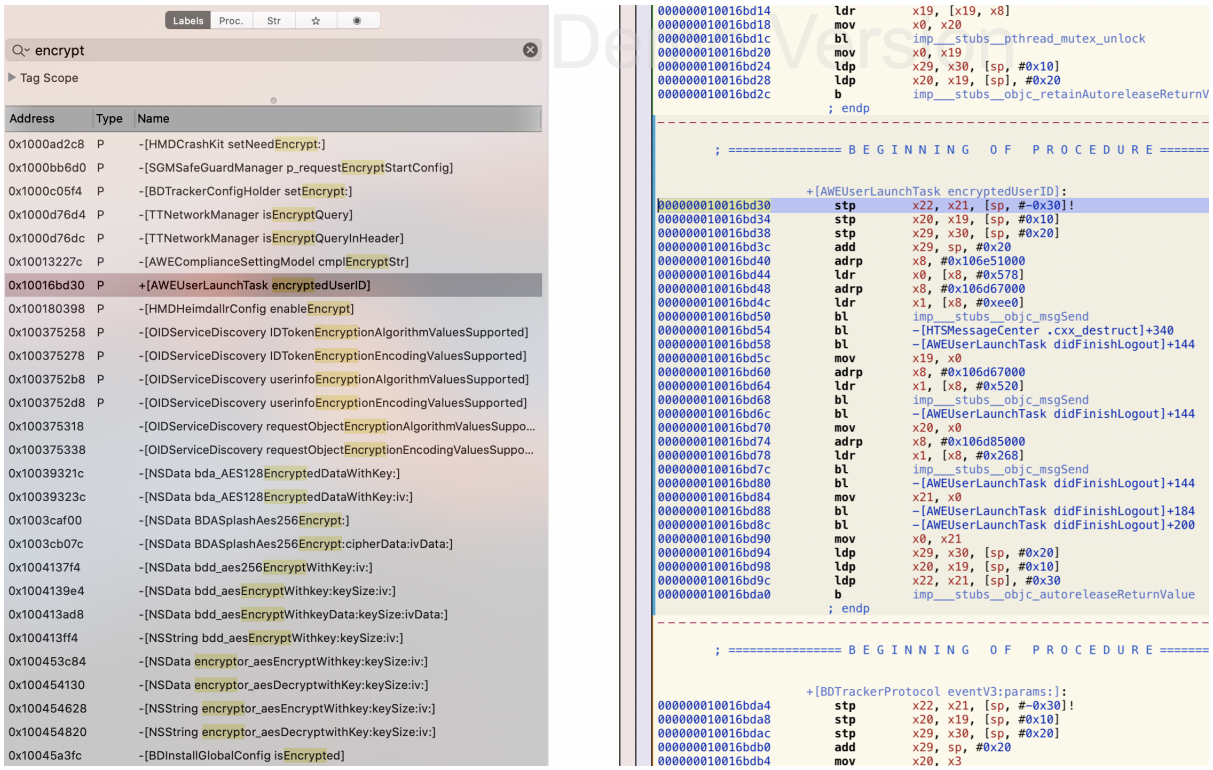


Figure 3.5: Hopper: Disassembling *TikTok* and searching for an 'encrypt' methods

The screenshot above shows an example of the operation of *Hopper*. In this case, the *TikTok* app has been disassembled. In attempt to understand how it encrypts the bodies of its requests, a user can search for the string 'encrypt' (left) and inspect app functions and classes that match this string. Hopper auto-generates C pseudo code from the assembly functions, which the user can inspect to build a better understanding of how the app actually functions. This can greatly aid the process of reverse engineering.

CHAPTER 4

Evaluation

Having established a successful model to perform privacy analysis, the internet traffic for specific apps were targeted and examined. As outlined in the introduction to this work, focus was firstly placed upon background connections that occur to Apple servers and exactly what they transmit.

Connections were logged and discovered through use of *mitmproxy*, with *SSL Killswitch 2* running on the device to unpin SSL certificates. In the case further SSL unpinning was required, a custom unpinning script was injected using *Frida*. It is important to note that of the majority of connections displayed only a subset of the data is shown. Omitted data is either uninteresting, repetitive or does not add to discussion in any way. At various points throughout the analysis process, *Frida* and *Hopper Disassembler* were also used to dive deeper into the underlying functionality of a process, in cases where extra efforts were required to gain access to an app and its data.

4.1 Apple

A wide variety of connections occur in the background of an iPhone's normal operation. These relate to different services required by the iPhone and are broken up in the following sections, based on the overall service they are known or deduced to be part of.

a. Apple Identifiers & Sensitive Information

iPhones use a variety of different identifiers for different purposes associated with the user and their device. Apple uses these to identify the device or the user regularly across multiple different Apple services. Some identifiers are unique and persistent, such as UDID or Serial number, and some refresh periodically like request or session identifiers which are sometimes used to identify a user's connection session with a server. A summary of the most important identifiers is provided as follows;

- UDID - Unique Device Identifier
- Serial number - Device serial number
- DSID - Digital Services Identifier
- IMEI - International Mobile Equipment Identity
- MEID - Mobile Equipment Identity (IMEI with hexadecimal digits)
- x-mme-device-id - Unique device identifier
- x-apple-i-md - Unique device identifier
- APS-Token - Apple Push Notification Service token

Of course, more standard forms of information about the user like name, address and gender are also considered as sensitive information as outlined in section 2.3.

b. iCloud

iCloud is Apple's cloud platform, which the iPhone will interact with when the user signs into their Apple account on the device. It manages integration of settings and preferences from other Apple devices, and provides a cloud drive for online file storage. There are several regular connections to iCloud services that carry device data, which are related to configuration, syncing and saving of settings on iCloud servers.

```
POST https://gateway.icloud.com/ckdatabase/api/client/subscription/create

POST https://gateway.icloud.com/ckdatabase/api/client/subscription/retrieve

POST https://gateway.icloud.com/ckdatabase/api/client/record/sync

POST https://gateway.icloud.com/ckdatabase/api/client/record/save
```

Each of the above connections use the same identifying request information, specified as follows;

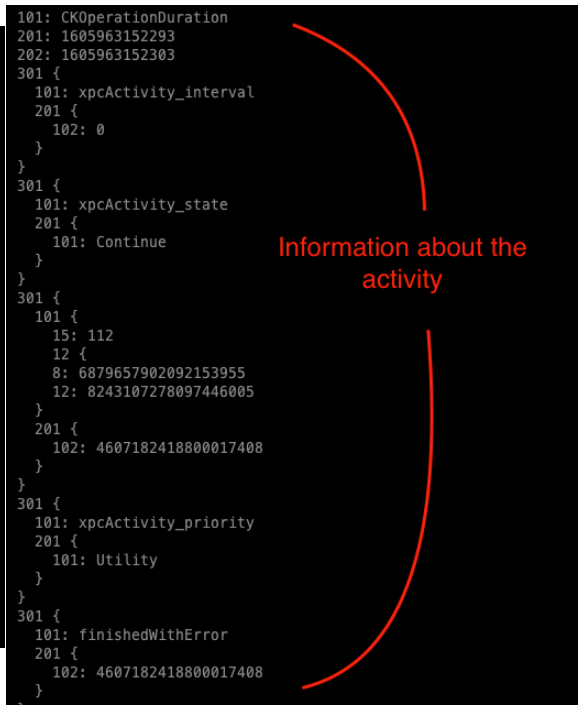
```
Request Headers:
    "x-apple-i-md-m": "GHY...6o2"
    "x-apple-i-md": "AAAABQ...Aw=="
Request Body:
    "udid": "54b...5ba"
    "serialNumber": "F4G...C6H"
```

While carrying out slightly different functions, the above connections transmit the same sensitive device data, the main two forms being UDID and serial number, which is sensitive device information. They mainly serve to configure and update iCloud endpoints on the user's status, transmitting nothing overly interesting aside from the sensitive identifiers. It is worth noting the frequency of these requests, which can occur multiple times in an hour.

Of course, there are many other iCloud connections catering to configuration of settings, and also interaction with the user's cloud drive, however they generally do not contain anything unexpected or specifically interesting. iCloud does however collect metrics about the manner in which iCloud services are utilised. Multiple cloud-related processes run on the iPhone during its normal operation. The below connection request body appears to send information about these different cloud related processes, and some metrics information like if there were any errors, their duration, their identifier etc. In the example pictured in figure 4.1, the request body is shown. Some device information is specified like model and OS version, and then some information about the *'com.apple.syncdefaultsd'* process (which is the process being reported on in this case) is specified. Further into the body of the request, information about the process activity like its duration (CKOperationDuration), what error it finished with (finishedWithError), its priority (xpcActivity_priority) etc. are specified.

```
POST https://metrics.icloud.com/c2
```

(a) Device information and activities being recorded.



(b) Information about activities

Figure 4.1: Body of POST request to /c2, displaying activity reporting.

This information is assumably collected for analytical purposes, allowing Apple to analyse the behaviour of these processes. Apple manages syncing of information between devices registered with the same Apple account through its *p33-keyvalueservice.icloud.com* server, which hosts the */sync* endpoint. A base64 encoded data frame is sent here, decoded to show the sent information in the request body as follows.

```
POST https://p33-keyvalueservice.icloud.com/sync
Request Headers:
    authorisation:  Basic MTc....T09
        X-Apple-I-MD-M: GHY...6o2
    X-Apple-I-MD: AAAA...AAAw==

Request Body:
    DeviceGestalt1k0
    ModelNameiPhone0
    OSVersion18A83950
    ...
    ComputerNameThomas's iPhone // device name
    0|Octagon
```

```
    PublicSigningKey....
    SerialNumberF4G... // iPhone serial number
    ...
    MacBook Pro0MessageProtocol
    Version0 ComputerName Mac-f01...
    SerialNumberC02... // Macbook serial number
```

Identifying some of the sensitive IDs observed here;

- iPhone Serial number = FG4...
- Macbook serial number = C02...

Since the Apple account signed in on the iPhone also is registered to a Macbook, the serial numbers of both devices are sent to this endpoint along with some other public keys used by the devices relating to Apple services. This allows Apple the capability to relate activities carried out by the same account across different devices. In this manner, any activity carried out by a user on any of their Apple devices will be linked under the same Apple account.

c. Advertising services

Multiple connections can occur to Apple advertising servers, some of which send personal information about the user. The connection below specifies the user's DSID and retrieves information about the user, most interestingly the user's eircode, year of birth, and gender alongside some other less important device information.

```
POST https://partiality.itunes.apple.com/WebObjects/MZPartiality.woa/wa/IAD/segment
Request Body:
        "dsid": "173...836"

Response Body:
    ...
    2: "A84 ..." (eircode)
    3: "1997" (year of birth)
    48: "IRL"
    ...
```

As of this connection, this data is linked with the DSID. Having retrieved this data successfully, it is then sent onwards to the *'cf.iadsdk'* server in the following connection.

```
POST https://cf.iadsdk/apple.com/adserver/2.6/segment/update
Request Body:
```

```
...
2: "A84..." (eircode)
3: "1997" (year of birth)
48: "IRL"
...
```

It is important to note here that this functionality is enabled due to the fact the device is configured to enable the *'Personalised Ads'* setting in the device's Privacy settings, which is enabled by default for this device. Indeed, examination of Apple's advertising targeting information in the device settings (Fig. 4.2) and support site (38) outlines why this data is collected for advertising purposes. This section also explains how the user is assigned to 'Segments' or 'Content Categories' based on how they have interacted with some of Apple's other services, essentially attempting to build a profile of the user's interests and use that as a basis for targeted advertising. This information is assumably also transmitted to the */segment/update* endpoint, but is encoded or encrypted in a fashion that is not readable.

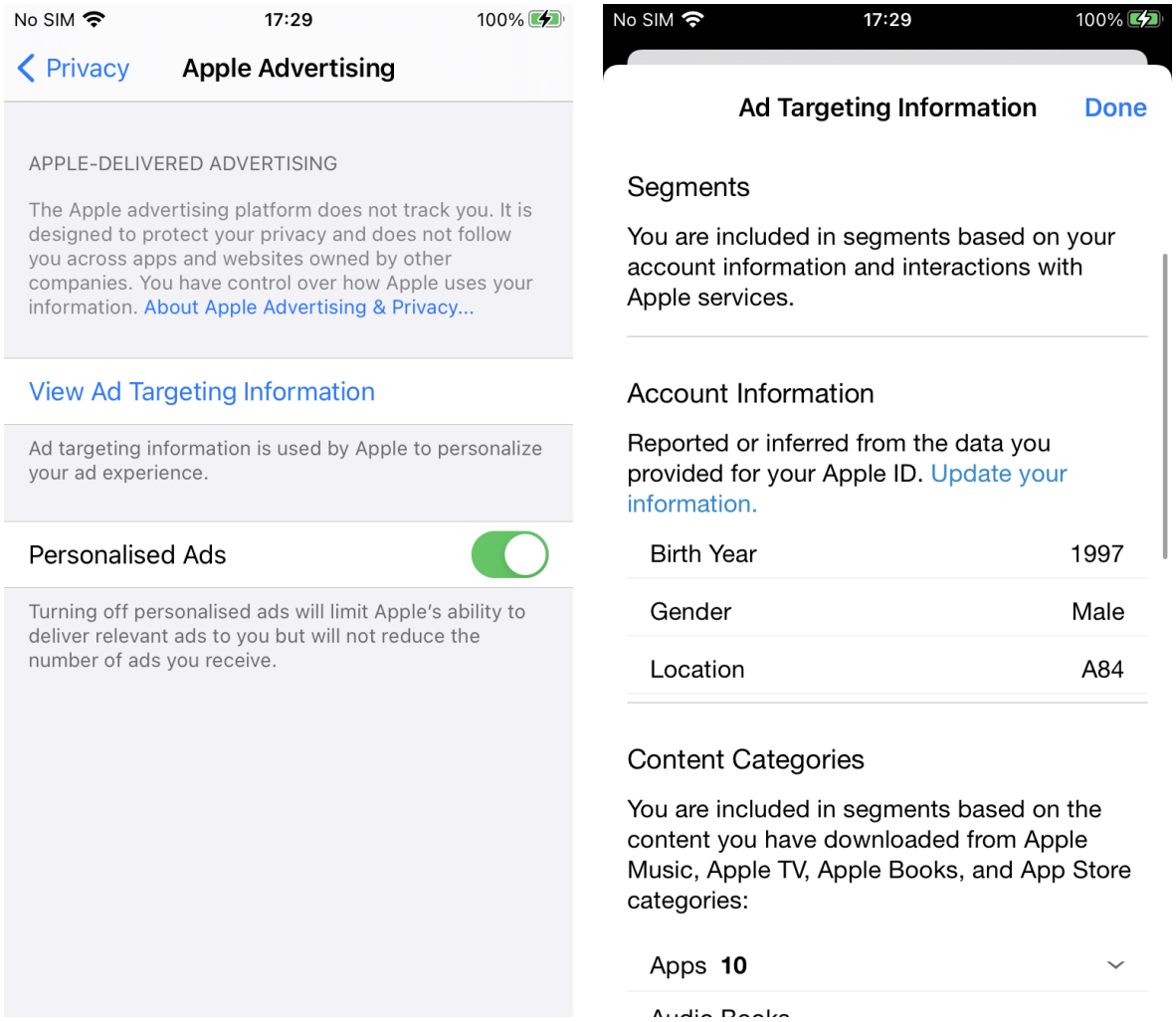


Figure 4.2: Apple Advertising settings and information

The iPhone will also occasionally send updates the the */attribution* endpoint when using an application. Advertising attribution refers to the process whereby advertisers can gauge how effective certain ads are, and find out what exact ads are having the most desirable effects. While the request body of the connection to this endpoint could not be decoded, it appears when a new app is opened some information is logged about it at this server. It may be the case nothing interesting is being transmitted in the encrypted body here, nonetheless, the connection occurs.

```
POST https://ca.iadsdk.apple.com/adserver/2.6/attribution
Request Body:
    ...x14\xc8\x98\xe8"\x0ecom.airbnb.app)\x00\x00\x80\xc5...
```

Apple assigns users an ad-related identifier which is commonly referred to in apps as *'advertising_id'* or *'idfa'*. *'IDFA'* stands for 'Identifier for Advertisers', and acts exactly as such. This ID serves as a way for application vendors to serve relevant ads to the user, which they acquire information about from Apple ad servers. The presence of the

same ID was verified in several different applications. While referred to differently across the applications below (even as '*hardware_id*' in the case of *Airbnb*), the identifier is partially denoted as '7F7...059' in the request bodies observed.

Instagram

```
POST https://graph.facebook.com/v9.0/12..14/activities
Request Body:
    "advertiser_id": "7F7...059",
    "advertiser_id_collection_enabled": "1",
    "anon_id": "XZ12...A065",
    "application_tracking_enabled": "1",
```

TikTok

```
POST https://api16-normal-c-useast1a.tiktokv.com/api/ad/splash/musical_ly/v15
Request Params:
    "idfa": "7F7...059"
```

Airbnb

```
POST https://api2.branch.io/v1/install
Request Params:
    "hardware_id": "7F7...059",
    "hardware_id_type": "idfa",
```

As can be seen, the ID is used across multiple different apps, which provides the basis for tracking a user across all of these applications using this identifier. In the case of *TikTok*, which is discussed at length later in this report, the *IDFA* is included as a parameter in a large chunk of the app's requests. This makes it even easier to attribute all activity made by the user in *TikTok* to this ID. The ID does however refresh on a regular basis, and while the exact interval was not determined it could be estimated to be at least once weekly. Furthermore, by configuring the iPhone's settings to disable cross-app tracking, is is observed that this sets the *IDFA* to a sequence of zeroes (0000-....-0000), by way of the */update* endpoint hosted at *iadsdk.apple.com*.

```
POST https://iadsdk.apple.com/adserver/2.6/segment/update
Request Params:
    ...00000000-0000-0000-0000-000000000000...
```

After this connection, applications cannot use the *IDFA* to pull advertising information from Apple servers. Unfortunately if they have already acquired that information, it is

entirely possible they can store it themselves.

d. App Store (iTunes)

Connections occur to App Store or iTunes related endpoints not only occur while using the App Store, but also when the device is idle. It should be noted that the two services (the App Store and iTunes) are often regarded as part of the same overall Apple service. The first connection, listed below, clearly sends a lot of sensitive information about the device in a connection appearing as being related to 'checking the app download queue'. Among the data sent, are the device serial number along with the UDID, along with some other session & application identifiers.

```
POST https://p30-buy.itunes.apple.com/Web/.../inAppCheckDownloadQueue?guid={udid}

Request Body:
    <dict>
        <key>appAdamId</key>
        <string>1505....357</string>
        <key>appDsid</key>
        <string>4qPdiWN....aMy8n6ug==</string>
        <key>appExtVrsId</key>
        <string>83...500</string>
        <key>bid</key>
        <string>in.nic.arogyaSetu</string>
        <key>bvrs</key>
        <string>16</string>
        <key>guid</key>
        <string>54b5......95ba</string>
        <key>serialNumber</key>
        <string>F4G.....C6H</string>
        <key>vid</key>
        <string>D8F4BD51-.....-.....AB0B-8B202E7B3682</string>
    </dict>
```

The App Store retrieves information it needs like remaining device capacity and screen type by way of the */xp_amp_appstore* endpoint. The user's DSID is also sent here. Of course the App Store can use this to determine whether of not the user's device is capable of downloading new applications.

```
POST https://xp.apple.com/report/2/xp_amp_appstore
Request Body:
```

```
    "capacityDataAvailable": "42800"
    "capacityDisk": "61000",
    "connection": "WiFi",
    "dsId": "17...36",
```

e. Find my iPhone (FMIP)

The FMIP service is updated on the state of the device at it's */register* endpoint. In terms of identification, the aforementioned *aps_ token* is sent, along with UDID and Serial number.

```
POST https://p33-fmf.icloud.com/fmipservice/fmf/{dsid}/{udid}/register
Request Body:
        "aps-token": "403...A48",
        "batteryLevel": 0.949999988079071,
        "appleId": "tkelly2@tcd.ie
        "batteryStatus": "NotCharging",
        "buildVersion": "18A8395",
        "deviceClass": "iPhone",
        "deviceColor": "2",
        "deviceName": "Thomas's iPhone",
        "serialNumber": "F4G...C6H",
        "smlLS": true,
        "supportsForceTouch": true,
        "supportsNotifyV2": true,
        "timezone": "Europe/Dublin",
        "udid": "54b...5ba",
        "Nfc": true
        "passcodeConstraint": "simple",
        "passcodeIsSet": false,
```

FMIP stores a variety of information about the iPhone, along with some information that could be useful for recovering it in the case it is lost. The above connection is witnessed regularly, at least multiple times per day, sending information like battery level, device color and if the device has a passcode. All of this information is of course useful in the case the device is lost or stolen.

f. Idle connections

A host of very interesting connections occur during the iPhone's idle operation that are not specifically related to any of the above headings. They are discussed in turn below.

/dispatcher.arpc

Calls to the */dispatcher.arpc* endpoint are witnessed regularly, observed at least several times daily. They are identified with an identifier for the user in the request body (*080...D39*), which is assumably attained in a previous interaction with this server.

```
POST https://gsp-ssl.ls.apple.com/dispatcher.arpc
Request Body:
    ...
    08064802-26C9-....-.....-0E32D39
    ..
Response Body:
    ...
    A84 ...
    8: Mill...
    10: Deer...
    11: 1...
    12: 1... Deer...
    17: Deer...
    17: Mill...
        ...
        9: apple
        9: revgeo
        9: IE
```

It may not be immediately evident from the response body, but an estimate of the user's address is returned here. Interestingly however, the estimate is slightly incorrect, calculating the wrong house number in this case. This would indicate that the user address has not been provided to this server by the user, but calculated in some manner by the back-end. The response also mentions 'revgeo' in multiple places, which is often used to refer to the concept of Reverse Geocoding. This is a technique used to acquire address information such as street number or city district from a set of provided coordinates. Apple offers an Objective-C class for this exact purpose, called *CLGeocoder* (40). While this might explain how and what this interaction with the */dispatcher.arpc* endpoint is doing, as for exactly why it is happening, only Apple knows.

/grandslam

Another interesting Apple connection is observed in the occurrence of the 'grandslam' connection, which sends sensitive device information like IMEI and MEID identifiers, which are hardware identifiers unique to the device. The response contains a 'liveness' key which is valued at 8,6400,000. If this is assumed to be milliseconds, corresponds to 24 hours, which might indicate the occurrence of this connection every 24 hours - however this could not be verified. The request also denotes other information like if the phone is using a sim card. The key-value pair labelled *'pktn'* in the request body is the same value as the previously witnessed *'aps_token'*. With all of this in mind, it is evident that the *'grandslam'* connection appears to be a large dump of sensitive identifiers to this *gsas.apple.com* server. The purpose of this update is not discernable from the data.

```
POST https://gsas.apple.com/grandslam/GsService/postdata
Request Header:
    "x-mme-device-id": "54b...5ba" // UDID
    "x-apple-I-srl-no": "F4G...C6H", // Serial Number
    "X-Apple-I-MD-M":  ...
    "X-Apple-I-MD": ...

Request Body:
    <key>imei</key>
    <string>356.......03</string>
    <key>meid</key>
    <string>35......20</string>
    <key>inUse</key>
    <integer>0</integer>
    ...
    <key>physicalSim</key>
    <true/>
    <key>slotID</key>
    <integer>1</integer>
    <key>ptkn</key>
    <string>403......A48</string>
```

/pbcwloc

Drawing attention to another miscellaneous Apple connection, traffic to */pbcwloc* appears to be quite rare. Decoding the request body using a decoding script add-on for *mitmproxy* (**Appendix A)**, the contents can be revealed to send a list of MAC addresses.

```
POST https://gsp10-ssl.apple.com/hcy/pbcwloc
Request Body:
    ...
    2c:91:ab:4c:..:..
    ...
    16:0a:c5:b5..:..
    ...
    76:a7:ea:0b:..:..
    ...
```

Further investigating the nature of these MAC addresses, the *arp -a* command can be used to view IP and MAC addresses of other devices connected to the local network being used. After cross-examining some of these IP addresses with the ones sent in the above connection, it could be argued the function of this endpoint is to gather the MAC addresses of devices connected to the same network as the iPhone, allowing the server to build a social graph of these devices. It is important to point out that MAC address is considered personal data by the GDPR, as it can be used to identify a person when combined with other data.

$ arp -a

```
fritz.box ... at 2c:81:ab:4c:..:.. on en0 ifscope ...
amazon ... at 14:a:c5:b5:..:.. on en0 ifscope ...
amazon ... at 74:a7:ea:b:..:.. on en0 ifscope ...
...
```

These locally observed MAC addresses can be matched with those sent in the body of the above connection. Some small discrepancy is noted between some of the values of the MAC addresses, however this is assumed as being down to variations in how the content is encrypted by the device or decrypted in the analysis process. Nonetheless, the need for collection of such data is undoubtedly questionable.

g. Maximising Privacy

As initially specified in the device configuration (section 2.1), the iPhone was initially set up to allow optional tracking behaviour. This section aims to quickly explore the question, 'How privately can an iPhone be used?'. Settings were configured as designated below.

While this configuration denies the user a lot of the services that make iPhones unique, it outlines the most 'private' way to use the device. Internet traffic is greatly reduced when such an effort is made.

Some of the connections witnessed upon disabling or signing out from the above

Setting	Configuration
Network Connection	WiFi
Location	Disabled
Cross-app Tracking	Disabled
Analytics	Disabled
Apple Advertising	Disabled
iCloud	Signed-out
Sim Card	None

Table 4.1: Summary of device settings

services are displayed below.

Ad Tracking

```
POST https://iadsdk/adserver/2.6/optout/optout_optin
Request Body: {encoded}

POST https://iadsdk/adserver/2.6/segment/update
Request Headers:
        limitAdTracking: true
```

After this exchange, no further interactions with the ad server have been observed, and the previously discussed *idfa* value is set to a string of zeroes.

Signing out of iCloud and FMIP

While signing out of iCloud in the device settings, the following connections are spawned. Firstly, a POST request to the */grandslam* endpoint sends a collection of sensitive IDs, including the DSID which is denoted as *'X-Apple-I-RINFO* in the first connection below. Other identifiers sent include UDID, Serial number and *aps_token* (ptkn).

```
POST https://gsa.apple.com/grandslam/GsService2
Request Body:
        <key>X-Apple-I-MD</key>
        <string>AAAABQA...VAAAAAw==</string>
        <key>X-Apple-I-MD-M</key>
        <string>GHYBKVIm...+fxCEfcDvl6o2</string>
        <key>X-Apple-I-MD-RINFO</key>
        <string>17...76</string>
        <key>X-Apple-I-Request-UUID</key>
        <string>FA0924......D7EA033</string>
        <key>X-Apple-I-SRL-NO</key>
        <string>F4...6H</string>
```

```
<key>X-Apple-Password-Only</key>
<true/>
<key>X-Mme-Device-Id</key>
<string>54b5...95ba</string>
<key>ptkn</key>
<string>403...A48</string>
```

The iPhone deregisters from the iCloud services in a flurry of several connections to different Apple services, listed in **Appendix D**. They once again send a variety of sensitive identifiers to endpoints like *'/deregisterDevice*. The following connection logs the user out of FMIP's service.

```
POST https://buy.itunes.apple.com/WebObjects/MZFinance.woa/wa/logout
Request Headers:
            X-apple-i-md-d: GH...6o2
            X-apple-i-md: AAA..w==
Request Body:
            udid: 54b....95ba
            imei: 356...203
            meid: 356...20
            serialNumber: F4G...H
```

After the appropriate settings are disabled and iCloud finishes its deregistration procedures, a great reduction in the device traffic is witnessed. A vast majority of the connections discussed in the previous sections do not occur under these settings, however the functionality of the device is greatly hindered. For example, without signing in through your iCloud/Apple account, apps cannot be downloaded from the app store, neither can any other iCloud functionality such as iCloud Drive (Files app) or Mail can be used. While a reduction in traffic was witnessed, more extensive analysis would need to be conducted to examine any further interaction with Apple servers.

h. Observations

The nature of the connections to a wide range of different Apple services means that most of the sensitive identifiers outlined at the start of this section (4.1.a) can be linked to one another across different services. In a more idyllic approach, each of these services would use specific identifiers which refresh after a given time period, lowering the chances of a user's identifiers being hijacked in a security breach. This implementation would however be quite costly and require more resources to store data for different services. In the case of some requests, it is entirely unavoidable to use a unique device identifier like UDID to ensure the user's identity, however inclusion of two

sensitive IDs, as UDID and serial number are often witnessed in company appears unnecessary. Furthermore, the constant updates of these sensitive identifiers appear unnecessary when they could just be stored securely on Apple servers - which they are likely already.

The purpose of some of the connections discussed above might also prompt questions as to their necessity. Examples of such connections might be that of the */pbcwloc* endpoint (subsection 4.1.f). This connection allows the server to build a social graph of MAC addresses on the same network as the iPhone in question, which does not seem necessary at all for an iPhone's normal operation. As a further example, the dump of information carried out by the *'grandslam'* endpoint begs the question as to why this needs to happen regularly.

While its purpose is obvious, the usage of the advertising identifier IDFA is not ideal, in the sense that multiple applications have access to it and can use it how they please. As was shown, the presence of this identifier in the requests of multiple different applications means it provides the basis whereby the user's behaviour can be linked on these different platforms. A more ideal approach might include storing a different IDFA for different applications, to eliminate the basis for cross app tracking using this identifier.

As discussed in the previous section, iPhone's do provide a variety of settings to allow the user to decrease their digital fingerprint when using their device. Settings such as configuration of analytics, ad tracking and location allow the user to control their preferences in terms of their privacy to a large extent, although not entirely removing tracking. In the case a user wants to minimise their online presence, Apple does make it possible to limit collection of data, leaving the user with functionality to make calls and SMS messaging. This however greatly limits services which make use of a smartphone like this worthwhile.

4.2 COVID Tracker Ireland

COVID Tracker Ireland is the Irish health service (HSE) contact-tracing app for COVID-19. It was first rolled out in June 2020 to combat the spread of the virus, and garnered roughly 2.4 million users as of February 2021.

a. Interactions with the app

The interactions taken with the app which produced the observed internet traffic are firstly outlined. After installing, the app was opened and a privacy information section is displayed to the user. Continuing through this, consent can be given to allow the

collection of app metrics, which are used to improve the app. At this point, the user is asked if they would like to enable contact tracing, turning on Bluetooth and enabling the Exposure Notification Service (ENS). With app metrics and ENS approved, the user is asked for their phone number. This is optional, although for the purposes of this work all optional functions were given consent and a phone number was provided. It was verified that changing stance on app metrics consent results in the expected behaviour of stopping the app recording metrics.

In the app, statistics relating to COVID-19 can be observed in the 'Updates' section, where data is pulled from back-end servers and displayed. The 'Check-In' functionality was also tested, whereby the user can specify some vague information about themselves (Gender, age-range, county and locality) and specify whether they are experiencing symptoms or not. The following sections expand on the internet traffic generated by such actions within this app.

b. Registration

Two connections are observed in the registration process for the app. The client firstly calls the */api/register* endpoint with a POST request. This endpoint returns a *nonce* identifier, which is a token that is typically used once for registration and then discarded. The subsequent connection uses a PUT request to the same endpoint, using the acquired nonce token along with a *'deviceVerificationPayload'* to uniquely identify the user for the back-end. The endpoint then returns a JWT token that lasts 12 hours which can be refreshed with a 'refreshToken'. This *token* acts as a unique identifier for the user while it is in use.

```
POST https://app.covidtracker.ie/api/register
Response Body:
    "nonce": "5229...0817"

PUT https://app.covidtracker.ie/api/register
Request Body:
    "deviceVerificationPayload": "AgAAA...0qT",
    "nonce": "5229...0817",

Response Body:
    "refreshToken": "eyJh...GdI",
    "token": "eyJh...Uz5C4",
```

Looking more specifically at the 'deviceVerificationPayload' parameter, it is generated through use of an external library called *react-native-ios11-devicecheck* (11). This library is interfaced with using React (JavaScript code) and executes Objective-C

functions to generate the ephemeral token to verify the device. The library uses the method *generateTokenWithCompletion* from the iOS system DCDevice (13) class to do this, which can be noted in the libraries' source code (14). After being sent as part of the PUT request to the */register* endpoint, the deviceVerificationPayload is subsequently sent onwards to an Apple endpoint *"http://api.devicecheck.apple.com/v1/validate_device_token"* (15), where it can be assumed that some checks occur to verify the device as a legitimate iOS device. This identifier is not processed any further by the *COVID Tracker* app's back-end, and does not appear to encode any sensitive information.

Following registration, the app has now acquired a *token* which it can use to authenticate its requests to the back-end, along with a *refreshToken* which is used to acquire a new *token* after expiry. These tokens are encoded as JSON Web Tokens (JWT), specified as a 'URL-safe means of representing claims to be transferred between two parties' in RFC 7519 (16). Decoding these tokens using the *jwt.io* online tool (17), their contents can be observed;

- token

```
{
    "id": "5cdf3203-e9be-....-....-ac05ca98831c",
    "iat": 1605606510, (17th Nov 2020, 09:48)
    "exp": 1605692910, (18th Nov 2020, 09:48)
    "aud": "com.hse.ftapp",
    "iss": "com.hse.ftapp"
}
```

- refreshToken

```
{
    "id": "5cdf3203-e9be-....-....-ac05ca98831c",
    "refresh": "0dfb4843b3ad889b574364.....9fd909ab8663b1df39fe3682f66f94",
    "iat": 1605606510, (17th Nov 2020, 09:48)
    "exp": 1921182510, (17th Nov 2030, 21:48)
    "aud": "com.hse.ftapp",
    "iss": "com.hse.ftapp"
}
```

As can be seen here, the tokens encode an 'id' for the user, which is a UUID (Unique User ID) given to the user by the server, which was verified not to persist across different installations of the app. Since the acquired token is used for authorisation in the header of the following app's API requests, these requests can be linked together.

Concerns around user privacy can arise here as the tokens can theoretically be used to link the identity of the user with his or her real world identity, by way of their phone number or IP address, as discussed in section 2.3 above. This is made possible because all requests to the back-end are linked together by the authorisation *token*, including the initial registration where the users phone number can be collected. Inspecting the source code (12), it can be observed that the *refreshToken* is used to set a new *token* each time one expires, which is every 24 hours. The *refreshToken* itself however is refreshed only every 10 years, and therefore establishes a link between the different tokens used in requests during that period. Notice that all requests listed use the same authorisation header value, which is the user's *token*.

Aside from phone number, the only other sensitive information obtainable by the app's back-end would be IP address or a device identifier. However the disclosure of personal information is made optional and the app's privacy policy (31) does state that the user's IP address is 'stripped at the server network layer on routing of the traffic to the application layer'. This would ensure the user cannot be identified by way of their IP address by the back-end servers. Regarding security tokens, they are deleted from the app upon signing out or deleting the app. They are however retained on the back-end for 60 days of inactivity before deleting. While this 'token-linking' does occur, the app makes several efforts to limit its threat to the user's privacy.

c. General Usage

Provided the user agrees, the */metrics* API endpoint caters to the collection of usage metrics.

```
POST https://app.covidtracker.ie/api/metrics
Request Headers:
    "authorisation": "Bearer eyJ...2pg",

Response Body:
    { "event": "TOKEN_RENEWAL",
    "os": "ios",
    "payload": "",
    "version": "1.0.4.95" }
```

The various metrics, taken from the source code are as follows;

```
    export enum METRIC_TYPES {
        CHECK_IN = "CHECK_IN",
        FORGET = "FORGET",
```

```
    TOKEN_RENEWAL = "TOKEN_RENEWAL",
    CALLBACK_OPTIN = "CALLBACK_OPTIN",
    LOG_ERROR = "LOG_ERROR"
}
```

Each request to the */metrics* endpoint notes the occurrence of one of the above events. This monitoring however can be refused by the user upon registration. The rest of the metric updates include information as follows;

- CHECK_IN

Signals when a user executes the 'Check-In' function on the app - which involves entering whether or not they have any symptoms.

```
{ "event": "CHECK_IN",
        "os": "ios",
        "payload": "",
        "version": "1.0.4.95" }
```

- FORGET

Occurs when the user leaves or deletes the app.

```
{ "event": "FORGET" ... }
```

- CALLBACK_OPTIN

If the user enters their phone number this event is logged as having occurred. It denotes whether or not the user has opted to be called back if they get marked as a close contact.

```
{ "event": "CALLBACK_OPTIN" ... }
```

- LOG_ERROR

This logs any type of error that might occur in the app and a description of the error. For example, an error might occur loading stats, checking-in, or uploading exposure keys;

```
{ "event": "LOG_ERROR" ... }
```

Coronavirus infection statistics for Ireland are acquired with a GET request to the */stats* endpoint.

```
GET https://app.covidtracker.ie/api/stats
Request Headers:
    "authorisation": "Bearer eyJ...2pg",
```

```
Response Body:
    "stats": [(covid stats)]
```

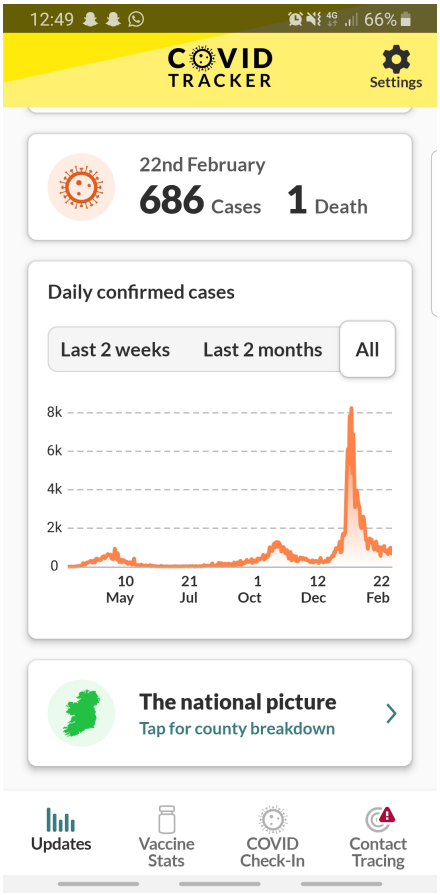


Figure 4.3: COVID Tracker App display of infection statistics.

Regarding the check-in function of the app, the user can enter some vague information about themselves and whether or not they have any symptoms of COVID-19. This function is of course optional.

```
POST https://app.covidtracker.ie/api/checkin
Request Headers:
            authorisation: "Bearer eyJ...2pg",

Request Body:
            "ageRange": "16-39",
            "data": [{ no symptoms }],
            "locality": "Meath, ...",
            "sex": "m,
            "Ok": true,
```

Published keys of infected people are retrieved with a call to the */exposures* endpoint.

The response returns a JSON object which denotes the path to different .zip files. The number observed in the path is a unix timestamp for the date and time the exposure update is retrieved.

```
GET https://app.covidtracker.ie/api/exposures?limit=6?since=12087
Request Headers:
    authorisation: "Bearer eyJ...2pg",

Request Parameters:
    limit: 6
    since: 12087, (1970)

Response Body:
[
    {
        "id": 12102,
        "path": "exposures/ie/1605952804798.zip"
    },
    {
        "id": 12117,
        "path": "exposures/ie/1605960005447.zip"
    }
]
```

The next connections retrieve the .zip files designated in the previous response. These files contain the diagnosis keys which are the keys randomly generated and uploaded by other user's devices who have tested positive. The app can then use the retrieved diagnosis keys to check against the keys of devices it has come within close contact of, therefore determining if the user is a close contact of someone who has COVID-19.

```
GET https://app.covidtracker.ie/api/data/exposures/ie/1605960005447.zip
Request Headers:
            authorisation: "Bearer eyJ...2pg",

Response: 1605960005447.zip
```

The below connection is dispatched to retrieve a new *token* after its expiry date is reached. The *refreshToken* is used to authenticate this request.

```
POST https://app.covidtracker.ie/api/refresh
Request Headers:
            "authorisation": "Bearer eyJh...GdI", (refreshToken),
```

```
Response Body:
            "token": "eyJ...6og", (new token)
```

d. Observations

User privacy is well respected in the Irish COVID app and measures are taken to protect user identities. All personal information required by the app is completely optional, including age, location and phone number. User data such as location and age are kept vague, through specification of an age-range and general locality, as opposed to a date of birth and exact GPS location. The metrics collected by the app are also not invasive, containing no sensitive information about the user and are disclosed fully in the app's privacy policy. The app's privacy policy also states that the user's IP address is stripped out at the network level of communication, and is not processed by the server's application level. While it is technically still attainable by the back-end servers, the policy assures that IP addresses will not be processed by the back-end. Regarding contact tracing functionality, the app uses Apple & Google's ENS, which ensures its user's identity cannot be identified in the event of a positive test result. Since this could not be tested without achieving a positive code test result, this functionality remained unexplored. There may be a debate to be had around how these measures affect the effectiveness of the contact-tracing app, however that is not relevant to the topic of this work.

In the interest of complete user privacy, concern must be drawn to the concept of token linking. Since the aforementioned *refreshToken* persists for 10 years, and is used to attain a new access token every 24 hours, all requests made to the app's backend can be theoretically linked together by the *refreshToken*, while requests within 24 hour periods are identified by the *token*. Due to the fact the user's IP address is stripped as previously mentioned, it cannot be used alongside the linked requests to track user location over time. Despite these measures, in the interest of ultimate user privacy, it is unnecessary to link requests together in this manner, and ideally could be avoided.

4.3 Aarogya Setu

Aarogya Setu is India's implementation of a contact-tracing app. While similar in its goals to *COVID Tracker Ireland*, it differs in both its methodology and concern for it's user's privacy. A particularly relevant point in the case of this app is the nature of the Indian Government's requirement for mandatory usage of the app (70) in May 2020. The Government has since backed-down on this policy, with only 83 million of 500 million Indian smartphone users downloading the app.

a. Interactions

Through the registration process of the app, consent was given the access location and Bluetooth data, alongside approval of the app's privacy policy. A One Time Passcode (OTP) is then used as verification to authenticate the user by way of an SMS message to their phone number. This was required to be an Indian phone number, and thus a temporary phone number site (20) was used in place of a real Indian phone. Upon successfully bypassing the OTP verification by submitting a temporary Indian phone number, entry was gained to the application.

The basic functionality of the app was tested similarly to the Irish implementation, this included updating statistics, checking-in with symptoms, and checking recent contacts. The app displays statistics for different regions in India as shown in figure 4.4.

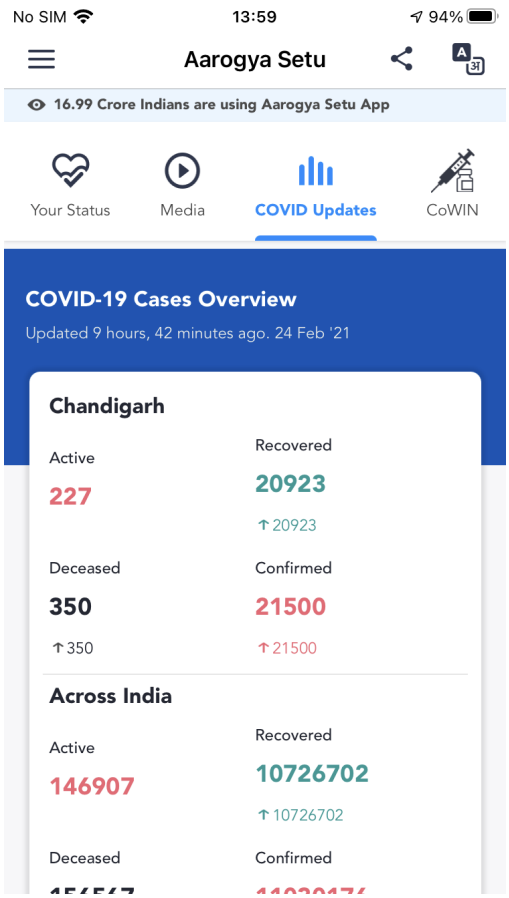


Figure 4.4: Aarogya Setu display of infection statistics.

The self assessment or check-in functionality was also tested multiple times. Self assessment results in a different color-coded response based on the COVID symptoms experienced by the user. A code green means the user is unlikely to be infected with COVID-19, while yellow and orange codes indicate there is moderate and high risks of infection respectively. In the case of the latter codes, or a positive test result, the user

will be prompted to upload their location and Bluetooth data. The user can also optionally - at any time - share their data with the government of India. Another function offered by the app is to view other app users within a set proximity. The app will then indicate whether or not the user has come within Bluetooth contact of any other users who tested positive, and if any those users feel unwell.

The registration process of the app firstly involves some configuration and retrieval of identifiers for interaction with Google's *Firebase* service (18). In this case, it is mainly used for analytics and logging purposes although *Firebase* offers many other functions.

b. Google Firebase connections

i | Registration

Upon initially opening the app after after installation, some connections occur related to configuration for a host related to device provisioning. The connection acquires an *android_id* and *security_token* that the device uses in some following requests to google services. These values are confirmed not to persist across different installations of the app, and are therefore app instance identifiers, along with other values in the response (shown below) like the *digest*.

```
POST https://device-provisioning.googleapis.com/checkin
Request Body:
    "model": "iPhone10,
    "os_version": IOS_14.1",
    "locale": "en_IE",
    "time_zone": "Europe/Dublin"

Response Body:
    "android_id": 55252......18528000,
    "device_data_version_info": "ABF...YPbw",
    "digest": "MH....Q==",
    "security_token": 6120.....277,
    "user_serial_number": 0,
```

The next connection observed is to the */installations* endpoint, which returns the token necessary to authenticate the app's connections to *Firebase*. The identifier labelled *'fid'* in these connections is referred to in Google *Firebase* documentation as *Firebase* installation ID (FID) (32). Due to the fact the *FID* is used to retrieve a new token, every request to refresh the JWT is linked together by the *FID* value, which is an app instance identifier. It does not appear to be refreshed for the duration of the app's

current instance, only changing if the app is deleted and reinstalled.

```
POST https://firebaseinstallations.googleapis.com/v1/projects/.../installations
Request Body:
    "appId": "1:6453....56042:ios:8ac8fed...2bf8c29cfe73",
    "authVersion": "FIS_v2",
    "fid": "eZK_QXxV1...kaMS0cL_H",
    "sdkVersion": "i:1.1.0"

Response Body:
    "authToken": {
        "expiresIn": "604800s",
        "token": "eyJh...6og",
    }
    "fid": "eZK_QXxV1....kaMS0cL_H",
    "name": "projects/6425235/installations/eZk_QXx...cL_H",
    "refreshToken": "2_b_Dl...Ne2",
```

The JWT token returned by the request is decoded and shown to contain the *FID* along with two other identifiers for the *Aarogya Setu* ap (*projectNumber & appId*).

```
    {
        "fid": "eZK_QXx...6mkaMS0cL_H",
        "projectNumber": 6453...6042,
        "exp": 1613133295,
        "appId": "1:645...756042:ios:8ac8f...f8c29cfe73"
    }
```

The *projectNumber* & *appId* identifiers are used to identify the *Aarogya Setu* application to *Firebase*, and are therefore persistent although not sensitive.

The final connection regarding registration for Google *Firebase* is shown below, which is a connection to register the user with the *'fcmtoken'* Google API. In the authorisation header of this connection, the aforementioned *android_ id* and *security_ token* identifiers are used to verify the user's identity. The JWT returned from */installations* above is also used as the *x-goog-firebase-installations-auth* header ("eyJh...6og").

```
POST https://fcmtoken.googleapis.com/register
Request Headers:
    "authorisation":   "AidLogin 5525257...28000:6120...42277"
    "x-goog-firebase-installations-auth": "eyJh...6og",

Response Body:
    "token": "eZK_..._CGa"
```

ii | General Usage

The below connection sends *FID* along with an app identifier and *DID* (discussed later) to identify itself along with some other information about the device like country code and platform version. The response contains some configuration settings information for the app, which includes *'launch_count_for_rating'* - i.e. after how many launches will the app ask the user for a rating, among others.

```
POST https://firebaseremoteconfig.googleapis.com/.../firebase:fetch?key=AIz...YFU
Request Body:
    "app_instance_id": "eZK_...._H"
    "app_id": "1:645345756042:ios:8ac8fed9ca2bf8c29cfe73"
    "country_code": "ie",
    "language_code": "en",
    "Platform_version": "14.1"

Response Body:
    {
        "appName": "in.nic.arogyaSetu",
        "entries": {
            "blur_location_meter_ios": "0",
            "exclude_battery_check_enabled": "true",
            "launch_count_for_rating": "20",
            "scan_poll_time_android": "60",
            "scan_poll_time_ios": "30"
        },
        "state": "UPDATE"
    }
```

After registration, a frequent connection observed is to the *app-measurement.com/a* endpoint, which sends metrics and usage information. The data is encoded in a protocol buffer (19) and contains timestamps of what screen controllers are used at certain times. This could be observed as monitoring what parts of the app the user is utilising, and how long they spend using them.

```
POST https://app-measurement.com/a
Request Body:
    {
        1 {
          1: _si
          3: 25058604070285130
        }
        1 {
```

```
      1: _sc
      2: Aarogya_Setu.HomeScreenViewController
    }
    1 {
      1: _sn
      2: webviewScreen
    }
    1 {
      1: _o
      2: auto
    }
    2: _ab
    3: 1608231964039 (December 17, 19:06)
    4: 1608229233465 (December 17, 18:20)
  }
```

c. Aarogya Setu connections

i | Registration

During the registration process the app requires an OTP for verification which is sent to the user's phone number. As mentioned, an Indian phone number was acquired through use of a temporary phone number website, as phone numbers outside India were unable to receive the OTP code. The connections below show the OTP verification process, noting that the user's phone number is sent, alongside an identifier labelled *'pt'*, which is a unique identifier which will be examined in more detail later.

```
POST https://fp.swaraksha.gov.in.com/generateOTP
Request Body:
    "primaryId" : "+917.....963"

Response:
    "message": "OTP sent"

POST https://fp.swaraksha.gov.in.com/validateOTP
Request Headers:
    "pt": "fc643....98f"

Request Body:
    "primaryId": "+917.....963"
    "passcode": "278123"

Response:
```

```
"auth_token": "eyJ....XOBo"
"refreshToken": "eyJ....cq3U"
```

Having successfully passed the OTP verification, the device now has an *auth_token* it can use to authenticate subsequent requests to the app's servers. The JWT is shown in its decoded state below, encapsulating the user's phone number and 'username' value which is a unique identifier for the user for their current instance of the app. It does not however persist across different installations.

- auth_token

```
{
    "exp": 1612614989, ( → 6th Feb, 12:36)
    "iat": 1612528589, ( → 5th Feb, 12:36)
    "sub": "+9174...963",
    "username": "25a...277"
}
```

- refresh_token

```
{
    "exp": 1613824589, ( → 20th Feb, 12:36)
    "iat": 1612528589, ( → 5th Feb, 12:36)
    "sub": "+9174...963",
    "username": "25a...277"
}
```

The *refresh_token* expires after a 15 day period, within which it is used to acquire new *auth_token*'s. This allows linking together of requests made within this period. However, even requests across 15 day periods can be linked due to the inclusion of the user's phone number encoded in both *auth_token* and *refresh_token*s. This can be used to link requests together, but is also personal information that theoretically could be used to track the user and reveal their identity, as has been discussed in previous sections. Another interesting observation is that the *'pt'* identifier witnessed in these connections actually persists across different installations of the app, even several weeks apart. This would mean the identifier is either unique to the iOS device being used, or is simply an identifier assigned to all iOS users of the app. Examining the source code (33), this value is revealed as being a *platformToken*, which is specified in the *keystore.properties* file in the application. The source code *readme.md* file denotes that this 'can be any UUID (unique user identifier)' and therefore its nature and what information it might contain is unclear. In the case it is an identifier unique to the user as the *readme.md* outlines, it would serve to link all of the user's requests together,

which really defeats the point of using refresh tokens.

The user is registered as of the below connection, identified by the *pt* identifier and *auth_ token* which stores their phone number and temporary username. This JWT ('eyJ...XOBo') which was received earlier from the */validateOtp* endpoint performs the authorisation for all requests made to the app's back-end. The *ft* identifier is also sent, which is the token received from *Firebase* registration. While denoted with *ft* in the below connection, the same value is referred to as *FID* in the *Firebase* connections. The endpoint returns a *'DID'*.

```
POST https://fp.swaraksha.gov.in.com/api/v1/users/register
Request Headers:
    "pt": "fc643...98f"
    "authorisation": "eyJ...XOBo"

Request Body:
    "ft" : "ezK...CGa"
    "is_bl_allowed": true,
    "is_bl_on": false,
    "is_loc_allowed": true,
    "is_loc_on": true,
    "n": "iOS User"

Response Body:
        "did": "ac...7"
```

The *DID* value observed here is an identifier which is later seen to be uploaded along with the user's Bluetooth scans and GPS location. It is a unique identifier for the user for their current instance of the app, however it refreshes after a new installation of the app. The app's Privacy Policy (34) states that 'the DiD will thereafter be used to identify you in all subsequent App related transactions and will be associated with any data or information uploaded from the App to the Server'.

ii | General Usage

The */users/fcm* endpoint appears to cater to the updating of the *FID* on *Aarogya Setu* servers. With the occurrence of this connection, *Aarogya Setu's* back-end can now identify what information *Firebase* stores on the user.

```
POST https://fp.swaraksha.gov.in.com/api/v1/users/fcm
Request Headers:
    "authorisation": "eyJ...XOBo",
```

```
    "pt": "fc643....98f"

Request Body:
    "ft" : "eZK_..._H",

Response Body:
    "did" : "acd...7",
```

A status update for the user is acquired by the app through the */users/status* endpoint. This informs the user of their health risks based on their symptoms, or if anyone in their proximity has tested positive for COVID-19. It also indicates whether or not you should quarantine, along with some other configuration options. The *'color'* field returned here designates the user's COVID-19 risk level, which ranges in order of increasing risk through green, yellow & orange. The endpoint also returns a '*p*' field, which is found to be representative of whether or not the user's GPS and Bluetooth data should be uploaded silently within the app based on their risk level. This is discussed in more detail later in this report.

```
POST https://fp.swaraksha.gov.in.com/api/v1/users/status
Request Headers:
    "authorisation": "eyJ...XOBo",
    "pt": "fc643...98f"

Response Body:
        {
    "did": "ac...7",
    "full_upload": "0",
    "meta": {
        "color": "green",
        "radius": 0,
        "target": "https://web.swaraksha.gov.in/ncv19/"
    },
    "p": 0,
    "proximity": 0,
    "s": "healthy",
    "self_assess_popup": 0,
    "self_assessment_status": "great",
    "show_form": false,
    "show_link": false,
    ...
}
```

The app attains information about other users of the app within a specified radius through the following connection. Exact coordinates of the requesting device are specified. The response contains information about nearby users like if they are unwell, infected or have carried out self-assessments.

```
POST https://webapi.swaraksha.gov.in/ncv19/nearby-stats/?dist=1km
Request Headers:
    "authorisation": "eyJ...XOBo",
    "pt": "fc643...98f"
    "Lat": 53... // latitude
    "Lon": -6... // longitude

Response Body:
    {
        "bluetoothPositive": 0,
        "infected": 0,
        "selfAsses": 0,
        "success": true,
        "unwell": 0,
        "usersNearBy": 1
    }
```

The self-assessment functionality is regarded as optional but is strongly encouraged at multiple points during the app's normal functioning. It sends symptom data and GPS coordinates.

```
POST https://fp.swaraksha.gov.in.com/api/v1/user/chat/data/
Request Headers:
    "pt": "fc643...98f"
    "auth_token": "eyJ...XOBo"

Request Body:
    {
        "confirm": "Ok",
        "lat": "53.....",
        "lon": "-6......",
        "status": "eligible",
        "symptom_past": "Lung disease, Hypertension, Diabetes, Heart Disease",
        "symptom_past_travel_social": "Travelled internationally in the last 14 days",
        "symptoms": "Difficulty in Breathing"
    }
```

The user is able to share their exact GPS location & Bluetooth scan data with the app's authority, which is the Government of India. They can do this willingly at any point while using the app, and will also be prompted to do so after taking a self assessment that indicates high likelihood of having Coronavirus. In any case, the data is sent to the endpoint as follows;

```
POST https://fp.swaraksha.gov.in.com/api/v3/users/data/
Request Headers:
    "pt": "fc643...98f"
    "auth_token": "eyJ...X0Bo"

Request Body:
    \x1f\x8b\x08\x00\x00\x0\.........dcF\x01\x00\x00
```

The content of the request body is encoded as a protobuf, and can be decoded with use of a python script (see **Appendix A**) developed to be used as an addon for *mitmproxy*. The script returns the request body contents below.

```
{
  "m": "",
  "d": "ac....7e7", // DID
  "upload_type": "tested_positive_consent",
  "data": [
    {
        "l": {
        "lat": 53,.....
        "lon": -6......
    },
    "ts": "1612528589",
    "dl": []
    },
    ...
    {
        "l": {
        "lat": 53.......
        "lon": -6.....
    },
    "ts": "1612528589",
    "dl": []
    }
  ]
}
```

When decoded, it is clear the data contains time-stamped exact GPS coordinates of the subject device. In the case Bluetooth close-contacts are detected, they are also sent here. This information can be collected by the app in three cases, which are all outlined in its privacy policy. The first case, is in which the user tests positive for COVID-19. The second, occurs when the user completes a self assessment that indicates them as being likely to have COVID-19. They will be prompted to upload this information, but are however afforded the option to 'do this later'. The third is whereby the user's color-coded risk status is set to yellow or orange, indicating moderate or high risk of contracting the virus, based on contacts or assessment. In this last case, if the user had not read the privacy policy, they would have no idea the app is tracking them in real-time as the app gives no clear indication of the occurrence of this upload.

The capability that allows this can be observed in the source code for the iOS app (21), where checks occur for the connection response key p and if it equals '1'. If this is the case a function called *uploadBluetoothScans()* is called, which does as its name might suggest, alongisde uploading time-stamped GPS coordinates. This p field was mentioned before under the */users/status* endpoint, as that is where it is acquired by the app from the back-end. Investigating this finding further, the response from this endpoint was intercepted making use of *mitmproxy*'s capabilities to intercept and modify connection's requests and responses (22). The p field returned by the */status* endpoint was modified to be '1' along with setting risk level to 'yellow'. Subsequently, the user's Bluetooth and GPS data is uploaded to the above endpoint, although there is no indication of this within the app.

d. Observations

When compared with the Irish *COVID Tracker* app, *Aarogya Setu* undoubtedly employs some approaches that could be deemed more unnecessary and invasive in terms of user privacy. Before the user has even passed OTP verification, traffic occurs setting the user up with Google's *Firebase* service. The simple fact that a third party service is used - while convenient for development - is not ideal. While there is nothing inherently *wrong* about this, it is always optimal to store a minimum amount of user data, in as little places as is possible. Furthermore, the app is not transparent about its use of this third party service - with no mention in its privacy policy - although this service does not collect any sensitive information. The app uses a variety of tokens to identify the user, which are outlined below.

- DiD - 'Unique digital ID' - Used to identify the user, and is sent with any data or information uploaded from the app to servers
- auth_token - A JWT used to authenticate the user with the app's back-end. This

token refreshes every 24hrs, and when decoded reveals the user's phone number and username identifier. New *auth_token's* are retrieved with the *refresh_token*.

- platformToken ('pt') - An identifier sent in almost every connection to the back-end. It persists across different installations of the app, which, if unique links all requests together.
- Phone number - The user's phone number, sent to servers during OTP verification and subsequently encoded in every authentication token used by the app.
- refresh_token - Used to acquire a new *auth_token*. The refresh token itself expires every 15 days
- username - UUID encoded in the *auth_token*

These tokens are used in important connections in different combinations. As previously discussed, this provides ample opportunity for linking of the app's requests to the back-end servers. Sensitive information like the user's phone number is encoded in the auth and refresh tokens, which easily allows for hypothetical real-time tracking of the user, as has been previously mentioned. Furthermore, the 'pt' or platformToken is used very consistently in connections, which the project's *Readme.md* states is a UUID although this could not be verified. This simply provides an extra unnecessary medium whereby the app's requests can be linked, and ideally would be avoided. The encoding of extra information like phone number in the JWTs could also surely be entirely avoided. While the *DID* is a useful identifier for the user's account, its usage could also be avoided by using one single identifier that could be encoded in the authorisation token, which is the approach taken by *COVID Tracker Ireland*.

In terms of the app's strategy for contact tracing, it uses exact GPS coordinates of the user. When compared with other contact tracing apps like Singapore's *TraceTogether* (35) or indeed the Irish contact-tracing app, this seems excessive. In the case of these applications, they use Bluetooth scanning to trace the spread of the virus between phone users, surmising that exact GPS location traces are not necessary for tracking the spread of infections. This approach to contact-tracing is much more respectful of user's privacy, and there is no evidence that it is less effective. Collecting GPS data here could be deemed excessive and unnecessary, with less accurate location readings serving perfectly well for collecting location-based COVID-19 statistics.

A more interesting finding of this work is that of the 'silent' uploading of user data based on their risk level. As discussed, this was revealed through examination of the source code, where code enabling the back-end to change the user's status of consenting to uploading Bluetooth scans. The capability was verified as previously stated, through modification of a server response which enabled the functionality. While the privacy

policy states that this does occur - and explicitly states that this will not occur if the user's risk level is 'green' - there is little indication to the user within the app that their location is being tracked in real time while their risk status is maintained at this level. In this manner, the app is tracking where the user is and at what time. When they are assigned a moderate (or high) risk level through close contact or self-assessment, all of this data is uploaded to the server without any obvious indication within the app.

Further examples of contrast with the Irish *COVID Tracker* app can be witnessed by simply comparing their privacy policies and handling of user data. It is clear from the privacy policy of the Irish *COVID Tracker* that user privacy is a core concern and consideration of the developers and steps have been taken to ensure best practices are adhered to. As it is recognised as sensitive information, the Irish *COVID Tracker*'s back-end strips the user's IP address at a network level, in an attempt to assure the user's privacy in this regard. The app also keeps vague information about the user. No such effort is made, nor are any other extra measures taken to obfuscate user data in *Aarogya Setu*, aside from the uploading of GPS and Bluetooth data.

4.4 Airbnb

Airbnb is an app which allows users to search and book travel accommodation. For this study, the app was simply signed-in, and some search & exploration functionality was tested, which are the core functions of the app and provide interesting topics for discussion. Other functions offered by the app include booking accommodation and communication with hosts, however these are not discussed as nothing immediately interesting was found to occur in those contexts.

a. Registration

After installation, the app attains some configuration information for the device such as currency and language settings. The request is authenticated with the *x-airbnb-api-key* and *x-airbnb-device-id* identifiers, which are persistent across different installations of the app. While the *x-airbnb-api-key* may be the same for ever user, the *x-airbnb-device-id* based on its name is assumed to be derived from device identifiers somehow. It is also important to note that this identifier persists even when signed-in with different accounts. Therefore, requests made to *Airbnb* services across different accounts are linked together by this *x-airbnb-device-id*, as it is regularly included as a header in request to *Airbnb* servers.

```
POST https://api.airbnb.com/v2/client_configs
Request Headers:
```

```
    "x-airbnb-api-key": "915...5b2"
    "x-airbnb-device-id": "23f...34b"

Response Body:
    "bot_detection_config": [
        "action_name_ios": "phoneOTP/ios",
        "endpoint": "v2/phone_one_time_passwords",
    ],
    "country_alpha2": "IE",
    "currency_settings": [
        {
            "code": "AED",
            ...
            "name": "United Arab Emirates Dirham",
        },
        ...
```

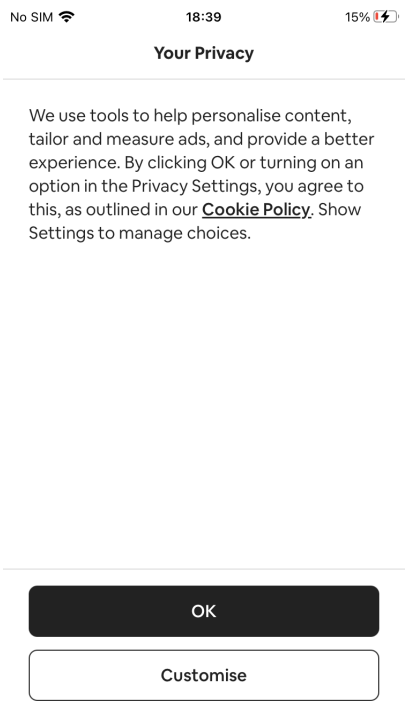


Figure 4.5: *Airbnb's* privacy dialog upon registration for the app

Several methods of signing-in are offered, namely Google, Apple, Facebook, Email or OTP verification through a phone number. Choosing the Google sign-in flow, a flurry of connections are observed to Google's services to authenticate by way of the OAuth service (54). This results in the user being provided with an *authCode*. The following connection uses this code to authenticate the user with *Airbnb's* back-end. It returns a unique identifier for the user, *'userId' (38...52)*. This identifier is specific to the instance

of an app, and therefore will not persist across different installations or accounts used in the app. It is mostly referred to as *'customerId'* or *'userId'*.

```
POST https://api.airbnb.com/v2/authentications
Request Headers:
    x-airbnb-device-id: 23f...34b
    x-airbnb-device-fingerprint:   MTQ...Dgt
    x-airbnb-api-key: 915...5b2
Request Body:
    "authCode": "4/0A...SkQ"

Response Body:
    "authAction": "LOGGED_IN",
    "accessToken": "ya29...Yp7",
    "authMethod": "GOOGLE",
    "userId": 38...52
    "token": "ee1...2ni"
```

b. Third Parties

Airbnb utilises several third party services for many different reasons. As mentioned above, several mediums of third-party sign-in are offered, although some of the more interesting services are used for functions like analytics, error monitoring, advertising and collecting customer information. It is also relevant to mention that all of these services are registered before the user has signed-in.

Facebook

One of the first connections for the app occurs to a Facebook endpoint. The response contains some language and configurations for Facebook's SDK and sign-in flow.

```
POST https://graph.facebook.com/v4.0
Request Body:
    "batch_app_id": "13..76" (Airbnb app identifier)

Response Body:
...
    "sdk_update_message": "Your Facebook SDK is out of date..."
    "recovery_message": "The server is temporarily busy, please try again.",
    "recovery_options": ["OK", "Cancel"]
```

The */activities* endpoint caters to more configuration for Facebook, sending some identifiers called *advertiser_id* & *anon_id*. The *advertiser_id* seen here is the Apple user's *IDFA*. Therefore, while the user is using *Airbnb*, and hasn't even signed in yet,

Facebook has attained their *IDFA* which they can use to pull advertising information from Apple's ad servers.

```
POST https://graph.facebook.com/v4.0/13..76/activities
Request Body:
    "advertiser_id": "14CB...4C2",
    "advertiser_tracking_enabled": "1",
    "anon_id": "XZF6...5D17",
    "application_tracking_enabled": "1",
```

reCaptcha

The app uses Google's *reCaptcha* (55) service to verify the user is not a robot.

```
POST https://www.recaptcha.net/recaptcha/api3/iosc
```

Geolocation

The app then registers and receives access tokens from the *geolocation.onetrust.com* authority as displayed in the below connection. OneTrust (59) is a company that specialises in privacy management for applications and web services. An article at CookiePro (which is a service run by OneTrust) (60) explains that the geolocation service is used to customise content based on a user's location. The article states that 'this is done by leveraging CloudFlare to perform a reverse IP lookup that will return the general location of the user'.

```
GET https://geolocation.onetrust.com/auth/v1/token/login/device/init
Request Headers:
    "signature": "eyJ...xXg"

Response Body:
    "access_token": "eyJ..QLE"
    "refresh_token": "eyJ...uSA"
    "tenantId": "fe1...164"
    "dsPortalId": "8b3...848"
    "preferenceCenterId": "2e6...a6d"
```

The 'signature' header observed above is a JWT token which is decoded to reveal the device-specific *Airbnb* identifier, which has been previously referenced as *x-airbnb-device-id*.

```
{
  "identifier": "23f6ca06f6823d7362ce0ec557c7f226a0b2e34b"
}
```

The response to this request contains an *access_token* which is displayed in its decoded state below, which encapsulates several identifiers for the *geolocation.onetrust* service,

including *x-airbnb-device-id*.

```
{
  "user_name": "23f...34b", // x-airbnb-device-id
  "scope": [
    "preferenceCenter"
  ],
  "tenantId": "fe...164",
  "exp": 1612529185,
  "preferenceCenterId": "2e6e...a6d",
  "authorities": [
    "USER"
  ],
  "jti": "040...adae",
  "dsPortalId": "8b3...848",
  "client\_id": "dsar"
}
```

Bugsnag

Another service is utilised, called *Bugsnag* (71) which handles error reporting and stability monitoring for applications. This service logs general information about the iPhone including if it is jailbroken or not, using an API key header and session identifier to manage the user's connection. No further connections were observed to the service, assumably because the no errors occurred during the app's operation.

```
POST https://sessions.bugsnag.com/
Request Header:
    bugsnag-api-key: 9d7...1f1

Request Body:
    "device": {
        "jailbroken": true,
        "modelNumber": "D20AP",
        "osVersion": 14.1
    }
    "sessions": [
        {
            "id": "7AC...689",
        }
    ]
```

mParticle

mParticle is service that collects customer data for the purposes of 'driving better

customer interactions' (61). It allows developers to connect their customers data to product analytics, targeted marketing and data warehousing tools through their API. Companies like Facebook, Google Ads, Snapchat, Twitter and Remerge have all integrated with this service. For example, leveraging their Facebook integration 'will enable Facebook to perform ad-tracking and attribution for your Facebook campaigns, by causing *mParticle* to forward app event data to Facebook' (62). As will be shown in later connections, *Airbnb* does send information about user behaviour like search history and device information to this service.

The connections refer to identifiers which have been previously identified as follows;

- 'ios_idfa' = *advertising_id* = IDFA
- 'other3' = *x-airbnb-device-id*

The user is intially registered with mParticle as follows:

```
POST https://identity.mparticle.com/v1/identify
Request Body:
    "device_application_stamp": "421...926",
    "ios_idfa": "14C...4C2",
    "ios_idfv": "863...B27",
    "other3": "23f...34b",
    "push_token": ""
```

Each time the *Airbnb* app is opened, *mParticle's* service receives an update on the device's identifiers in similar fashion to the previous registration endpoint *(/identify)*, however after this includes the *Airbnb userId* value.

```
POST https://identity.mparticle.com/v1/login
Request Body:
        "customerid": "38...52", // userId
        "device_application_stamp": "6AE6...6C3",
        "ios_idfa": "243...6A2", [AD ID]
        "ios_idfv": "1EB...A54",
        "other3": "23f...34b",
```

mParticle receives updates by way of the following endpoint, which lists device information and unique identifiers, along with some search history from the *Airbnb*, which in this case includes Galway City. As earlier connections to Facebook's *graph.facebook.com* server exhibit, Facebook also has access to the *'advertiser_id'* value. With both parties able to communicate with *mParticle* using this unique advertisement identifier, *Airbnb* can supply targeted material for advertising, which can

be subsequently displayed on the same user's Facebook profile. This is an example of how targeted advertising works.

```
POST https://natives dks.mparticle.com/v2/70...2e/events
Request Body:
    ...
    attrs: {
        "audience_type": "visitor",
        "checkin_date": "2021-02-28",
        "checkout_date": "2021-03-02",
        "city": "Galway",
        "country": "Ireland",
        "region": "County Galway",
        "search_string": "Galway City, Ireland",
        "user_bucket": "47",
        "user_group": "30"
    }
    ...
    "dn": "Thomas's iPhone",
    "aid": "14C...4C2" // IDFA (advertiser-id)
    "i": "23f...34b" // x-airbnb-device-id
    "i": "38...52" // userId
```

Sift science

Sift science is a company that provides a fraud prevention service. This service provides defense against attacks such as account takeover attempts, but also payment security mechanisms and defense against spam and scams. The app registers with this service as shown below. The service appears to send evidence of the device being jailbroken along with other information about the device's memory and model.

```
POST https://api3.siftscience.com/v3/accounts/529...709/mobile_events
Request Body:
    "cache_l1_dcache_size": 32768,
    "cache_l1_icache_size": 49152,
    "cache_l2_cache_size": 8388608,
    ...
    "cpu_logical_cpu_count": 6,
    "device_hardware_machine": "iPhone10,1",
    "device_hardware_model": "D20AP",
    ...
    "device_memory_size": 2070790144,
    "device_model": "iPhone",
    "device_name": "Thomas's iPhone",
```

```
    ...
    "evidence_files_present": [
        "/private/var/lib/apt",
        ...
        "/Applications/Cydia.app"
    ],
    "evidence_url_schemes_openable": [
        "cydia"
    ],
    "is_simulator": false,
```

Sift science acquires updates by way of the next connection, somewhat unusually acquiring information such as magnetic field data for the device, network addresses, and device orientation. The transmitted network addresses shown in the connection below take the form of IPv6 Link Local addressses (fe80::...), specified in RFC4291 (56). They are typically used for communication between nodes on an attached link (router). As to why they are required by *Sift science*, speculation could be cast on the fact these addresses could be used to build a social profile of other devices on the same network as the target device. The need for magnetic field data is more unusual. One study by C. Tejada et al. (57) shows how the magnetic sensors in modern mobile devices can be leveraged to determine a user's indoor location. While there is no evidence that this is how *Sift science* uses the data, the data is nevertheless transmitted as displayed below.

```
POST https://api3.siftscience.com/v3/accounts/529...709/mobile_events
Request Body:
    "device_orientation": "ui_device_orientation_face_up",
    "battery_level": 0.20000000298023224,
    "battery_state": "ui_device_battery_state_charging",
        "heading": {
            ...
            "raw_magnetic_field_x": 16.369277954101562,
            "raw_magnetic_field_y": -20.823753356933594,
            "raw_magnetic_field_z": -15.40911865234375,
            ...
        },
        "network_addresses": [
            "192.168.2.4",
            "fe80::..2c",
            "169....74",
            "fe80::...fc",
            "fe80::...fc",
```

```
            "fe80::...ef",
            "fe80::...a0"
        ],
        "time": 1616611923723
```

c. General Usage

Sessions of use within the app are opened and closed with connections to the */open* and */close* endpoints respectively. These connections are largely similar and send a variety of information, including identifiers, device information and configurations. They serve to update *Airbnb's* back-end about the device's status for every session of use within the app.

```
POST https://api2.branch.io/v1/open
Request Body:
    "ad_tracking_enabled": false,
    "apple_ad_attribution_checked": false,
    ...
    "cpu_type": "16777228",
    "device_carrier": "Carrier",
    "device_fingerprint_id": "886...585",
    ...
    "hardware_id": "14C...4C2",
    "hardware_id_type": "idfa",
    "identity_id": "886...579",
    ...
    "local_ip": "169.254.2.23",
    "screen_height": 1334,
    "screen_width": 750,
```

The app retrieves information about its users from the */accounts/me* endpoint. It is worth noting the name and birth date is not entered through the app but rather pulled from the Google account the app was signed-in with. The user is identified with their 'id', referred to before as *userId*.

```
POST https://api.airbnb.com/v2/accounts/me
Request Body:
    "currency": "EUR",
        "id": 38...52, // userId
        ...
        "is_facebook_connected": false,
        ...
        "required_steps": [
```

```
            "profile_picture",
            "phone_number"
        ],
        "user": {
            ...
            "all_active_phone_numbers": [],
            "birthdate": "1997-11-17",
            "country_of_residence": "IE",
            "email": "tkelly2@tcd.ie",
            "encrypted_id": "Z7c...MGA==",
            "first_name": "Thomas",
            "friends_count": 0,
            "government_id_dob": null,
```

Similarly, the user's notifications and inbox contents are pulled as follows.

```
GET https://api.airbnb/v2/notifications
Request Headers:
    x-airbnb-device-fingerprint:   MTQ...Dgt
    x-airbnb-device-id:            23f...34b
    x-airbnb-advertising-id:       14C...4C2

Response Body:
"notifications": [
        {
            "category": "promotion",
            "is_read": true,
            "is_seen": true,
            ...
            "name": "business_travel_offer",
            "priority": 1,
            "text": "Add your work email to unlock extra perks for business trips.",
        }
    ]

GET https://api.airbnb.com/v3/InboxPagesQueryGetInboxItems....
Request Headers:
    x-airbnb-device-fingerprint:   MTQ...Dgt
    x-airbnb-device-id:            23f...34b
    x-airbnb-advertising-id:       14C...4C2

Response Body:
```

```
    "inboxItems": []
    "archived": "0"
    "unread": "0"
```

/tracking/jitney is *Airbnb's* event logging service which is hosted at *www.airbnb.com*. This endpoint caters to regular updates about a variety of device-related data based on the occurence of different events. An example of some events might include *'native_measurement'*, which logs the invocation of different classes, another event called *'universal_session_start'* occurs when the app session begins. As a further example, an event called *'universal_api_response'* logs every API call made by the app and some information about it, such as status code, response size and response time.

```
POST https://www.airbnb.com/tracking/jitney/logging/messages
Request Body:
    [
        ...
        {
            "event_name": "universal_api_response",
            "host": "api.airbnb.com",
            "http_method": 2,
            "http_status_code": 200,
            "response_size": 103,
            "response_time_ms": 504,
            "server_total_time_ms": 77
        }
        ...
    ]
```

With regard to the core purpose of the app, which is to search for holiday experiences or accommodation, this is handled by the following connections, which exhibit example behaviour for searching for a stay in Galway, Ireland.

```
GET https://api.airbnb.com/v2/explore_tabs...&query=Galway%20City...
Request Headers:
    x-airbnb-device-fingerprint:   MWJ...zIt
    x-airbnb-device-id:            23f...34b
    x-airbnb-advertising-id:       14C...4C2

Request Parameters:
    adults: 2
    checkin: 2021-02-28
    checkout: 2021-03-02
```

```
Response Body:
    "lat": 53.27056400000001,
    "lng": -9.0566819,
    "location_shape": {
        "id": "93b59378-c597-49cc-a568-fabca2690ae3",
        "name": "Galway City"
    },
    "listings": [ {listings} ]
```

This request returns information about the searched location like its coordinates, and produces an array of listings to be shown to the user.

d. Observations

It is first worth noting the variety of identifiers used by *Airbnb* to identify a user, and pointing out that they are spread across a variety of different services. As before, the issue of linking requests by IDs is certainly present here, with the *x-airbnb-device-id* used quite frequently along with the *userId* and *IDFA*. Due to the fact each of the third party services typically use one or more of these identifiers in their requests, all requests to these services can theoretically be attributed to the same user.

In this way one of the most striking findings with *Airbnb* is the wide range of third party services used by the app, which might prompt questions about their necessity. For example, the need for usage of the *Geolocation* service appears altogether unnecessary, as the user's general location can be revealed from their IP address. For the app's purposes, attaining the users address also appears unnecessary. Also to be clear here, it is important to note no evidence was found in this work of the *Geolocation* service being used to determine user address, only that the user is registered with it when opening the app. Moving focus to the *Sift Science* service, a user might be surprised to know this service is monitoring their device data like the device name and cache size, but also collects evidence about whether their iPhone is jail broken or not. Perhaps most concerning is the fact the service collects magnetic field and local MAC network address data. The exact applications of this data could not be discerned, only that it is certainly being collected by the back-end. Speculating on the collection of such data, attention was previously drawn to the fact that local position can be determined with use of magnetic field data, and that network addresses of other devices on the same network as the iPhone can be used to build a profile of the network. Of course this is just speculation. In any case, the collection of such data appears unusual for a travel lodging app, and isn't specified explicitly in the app's privacy policy (63).

The most frequently observed third party connections were those that interfaced with *mParticle*. While there is nothing inherently wrong about use of a third party service, and the privacy policy does state that third party services *may* be used for advertising, this is not made abundantly clear to the user upon signing up to *Airbnb*. The privacy policy states 'We may use personal information to...' rather than 'We almost certainly will use your personal data to...'. The findings here are pretty clear as to how the advertising services are being utilised. *mParticle* collects user data from *Airbnb* like search history and makes it available to its partners, the prime example of which is Facebook. While a user might be under the impression that disabling Targeted Advertising in the iPhone's settings might prevent the occurrence of this process, due to the fact *mParticle* is a third party service, Apple cannot control what data it collects for advertising once the user has consented upon signing up for the app. While *Airbnb* does outline third party services are being used, it certainly isn't obviously transparent about what these services do, or what data they collect. Furthermore, the user isn't given an obvious option to opt-out of these services within the app. It is also worth stating that the abundance of third party services in this app is not ideal in and of itself, as due to the nature of data leaks and hacks in today's world, the more spread out user data is, the less secure it is.

4.5 TikTok

TikTok is a hugely popular social media app owned by the company *Bytedance*. It is essentially a platform for making and sharing short videos and has garnered approximately 689 million users since its launch in 2016. The app has drawn scrutiny regards its security and privacy, due to claims it is being used by the Chinese Government to spy on US citizens. Such was the concern for the application that former US president Donald Trump attempted to ban the application in 2020 (58). The following section attempts to address some of these concerns on the iOS platform, and analyse the app's handling of user data.

a. Gaining entry

TikTok immediately posed more challenges in terms of gaining access to the app and enabling the usage of the tools used to conduct analysis. The app uses a third-party library to perform certificate pinning, and therefore *Frida* needed to be utilised to attempt a bypass of these measures. After many unsuccessful attempts, it was eventually discovered by chance that the first time the app is opened after installation, certificate pinning does not occur. This is assumably because the first instance of the app connects to back-end servers without HTTPS, only pinning the certificates after

this first instance of the app. In this manner, the internet traffic from first instance of the app is fully observable using *mitmproxy*, which is a noteworthy finding in terms of security in and of itself. However, some specific connections included request bodies which were encrypted. In order to view the data being transmitted in these requests in an decrypted state, *Frida* was used to examine the data being sent before it was encrypted.

There was three specific sets of connections for which this was the case.

- Connections to the *Apps Flyer* service (65).
- Connections to *TikTok's '/app_log'* endpoint
- Traffic to the monitoring service, identified by the */monitor* endpoint.

The decryption issue was partly resolved with the injection of two *Frida* scripts (see **Appendix B & C**). The encryption methods these scripts target were discovered and examined using the *frida-trace* command. Both scripts essentially work by targeting the method provided by the user and intercepting its execution. They then print out the arguments passed and return value of the method. The usage of these scripts are outlined in more detail as they arise below.

b. Registration

Approximately 90 connections are observed when the app is firstly opened. When compared to other applications discussed in this work, this is quite a large amount. A vast majority of these connections acquire things for the app that are uninteresting, such as configuration settings and language strings. Among the first of the registration connections is to a */device_register* endpoint.

```
POST https://log-va.tiktokv.com/service/2/device_register/...
Request Header:
    x-ss-stub:              5EB...B93
    x-tt-trace-id:          00-6...6-01
    x-khronos:              1616768551
    x-gorgon:               840...d1e

Request Parameters:
    ...
    aid:                1233
    vid:                645...B34
    screen_width:       750
    openudid:           6e4...26f
    cdid:               6A0...984
```

```
    os_version:         14.1
    tz_name:            Europe/Dublin
    ...
    device_type:        iPhone10,1
    idfa:               ABA...B0B
    ...

Request Body:
    tc\x05\...\xc1\xa4

Response Body:
    {
        "device_id": 6950952705176094213,
        "device_id_str": "6950952705176094213",
        "install_id": 6950953156965287685,
        "install_id_str": "6950953156965287685",
        "new_user": 1,
        "server_time": 1618394913
    }
```

What is immediately noticeable about this connection, and indeed a majority of *TikTok's* requests is the amount of parameters included in the request. Those listed above are but a subsection of parameters sent, and are usually sent in requests to *TikTok's* servers. Due to the recurrence of the same volume of parameters in different connections, the parameters above are denoted by the *'general-params'* identifier henceforth for other connections. Alongside this, several other identifiers are often used in the request header. The ones included here *'x-ss-stub'*, *'x-tt-trace-id'* and *'x-gorgon'* appear to be IDs for the request, while the *'x-khronos'* value is a UNIX timestamp for when the request occurred. Examining the request parameters, *'idfa'* is the Apple *advertising_id* that has been discussed previously. Other IDs include *'vid'*, *'openudid'* and *'cdid'* are identifiers for the user which persist for the installation of the app. The endpoint returns some more identifiers, and denotes whether or not the user is new, based on their device. While it's label may indicate it as a persistent ID, the *'device_id'* passed here is unique for every installation of the device. It nonetheless serves to uniquely identify the user. The *'aid'* value 1233 is simply the app ID. The app proceeds by acquiring some configuration settings.

```
POST https://api16-normal-c-useast1a.tiktokv.com/tfe/api/request_combine/v1/...
Request Parameters: {general-params}

Response Body:
```

```
"aweme_will_stop_when_headphones_pulled": true
...
"shutter_sound_enable": false,
...
"tt_regions": "JP,HK,ID,MO,TW,KR,VN,TH,PH,MY,SG,KH,LA,MM,test",
"tt_use_settings_v3": true,
...
"log_in_order": "FACEBOOK,GOOGLE,TWITTER,INSTAGRAM,APPLE",
 "whatsapp_friend_invite_title": "Add WhatsApp friends here!"
```

Examining this connection response, a huge chunk of configuration information and language strings are acquired for the app, of which only a sample are shown above. This ranges from indicating whether or not the camera shutter sound is enabled, to a string for adding WhatsApp friends through the app.

```
POST https://api.snapkit.com/v1/config
Response Body:
    {
        "data": {
            "config": {
                "bitmojiLearnedSearch": {
                    "enabled": true
                },
                "skateConfig": {
                    "sampleRate": 0.01
                }
            }
        },
    }
```

Snap Kit (64) is a service run by social media company *Snapchat* that allows developers to harness tools offered by *Snapchat* like camera kits and custom emoji capabilities. The other social media service *TikTok* interacts with is of course Facebook, from which it acquires sign-in information in the same way as *Airbnb* from the *graph.facebook.com* server. Registration occurs for the *AppsFlyer* service, however the request contents could not be decrypted. *AppsFlyer* is discussed in more detail in section 4.5.e. The app uses *Firebase* for storing app information and metrics. As with *Aarogya Setu*, registration occurs for *Firebase* at */installations* as shown.

```
POST https://firebaseinstallations.googleapis.com/v1/projects/musically../installations/
Request Body:
    {
        "appId": "1:340331662088:ios:3c6c52c4762af402",
```

```
        "authVersion": "FIS_v2",
        "fid": "cFE...evE",
        "sdkVersion": "i:7.4.0"
    }
Response:
    {
    "authToken": {
        "expiresIn": "604800s",
        "token": "eyJ...MsQ"
    },
    "fid": "cFE...evE",
    "name": "projects/340331662088/installations/cFElpYSah0oBhiIoQ-LevE",
    "refreshToken": "2_U...Msy"
    }
```

The *FID* is seen once again here for *TikTok*, and allows the user to acquire an authentication token from *Firebase* which they can use for authenticating *Firebase* requests. Another server is used for what appears to be app performance monitoring, called *mon-va.tiktokv.com*. The first connection to this server registers user and device information with it.

```
POST https://mon-va.tiktokv.com/monitor/appmonitor/v2/batch_settings
Request Body:
    {
        ...
        "access": "WIFI",
        "aid": "1233",
        "carrier": "Carrier",
        "cdid": "755...43E",
        "device_id": "695...213",
        "device_performance_level": "2",
        "device_type": "iPhone10,1",
        ...
        "idfa": "7F7...059",
        "iid": "695...685",
        "install_id": "695...685",
        "is_env_abnormal": "1",
        "openudid": "ff4...14f",
        "os": "iOS",
        "region": "IE",
        "resolution": "750*1334",
        ...
```

```
        "vid": "AA5...D67"
    },
```

Information is also returned from the above endpoint, largely containing more configuration settings for how the device should interact with the monitoring server. After registering these different services, the app's server collects some information about the device to determine its region before prompting the user with sign in options.

```
POST https://api16-normal-c-useast1a.tiktokv.com/ttloc/submit
Request Params: {general-params}
Request Body:
    {
        "carrier_region": "",
        "locale": "en-IE",
        "mcc_mnc": "",
        "network_sim_region": "",
        "system_language": "en",
        "system_region": "IE"
    }
```

Having collected some evidence of the user's country, the app provides some sign in options including Google and Facebook. Choosing sign-in by way of Google Email address, Google's OAuth 2.0 sign in procedure is carried out.

```
POST https://accounts.google.com/_/signin/oauth
...
POST https://oauth2.googleapis.com/token
Response:
    "id_token": "eyJ...3DQ"
```

After the occurrence of the two connections above, the user is signed in through their Google account and given an *id_token* which is sent as *'access_token_secret'* to *TikTok's* back-end for verification.

```
GET https://api16-normal-c-useast1a.tiktokv.com/passport/auth/only_login/
Request Params:
    {general-params}
    "access_token_secret": "eyJ...3Dq"

Response Body:
    "platform_screen_name": "Thomas Kelly",
    "name": "Thomas Kelly89"
    "user_id": 690...297,
```

```
...
```

Decoding the JWT transmitted above, it can be observed how *TikTok* then has access to the user's Google account information.

```
{
    "iss": "https://accounts.google.com",
    ...
    "hd": "tcd.ie",
    "email": "tkelly2@tcd.ie",
    ...
    "given_name": "Thomas",
    "family_name": "Kelly",
    ...
}
```

Before admitting entry to the main part of the app, the user is asked to designate some of their interests, which are sent to the back-end as follows.

```
POST https://api16-normal-c-useast1a.tiktokv.com/aweme/v1/user/interest/select/
Request Parameters: {general-params}
Request Body:
    selectedInterestList:
    {"interest_list":
        [{"id":"Fitness & Health"},
        {"id":"OddlySatisfying"},
        {"id":"Travel"},
        {"id":"Comedy"}]}
    }
```

c. General usage

The main page of the app is an infinitely scrolling feed of videos, which the user can interact with by 'liking' or commenting on the video, among other things, like sharing. Any user can upload a video, who has their own profile page where all of their content is viewable. Regarding the main video feed, it is called the 'For You' page. Sets of videos are acquired for this feed ahead of scrolling by the following connection, with include an additional parameter designating the user's interests.

```
POST https://api16-normal-c-useast1a.tiktokv.com/aweme/v2/feed/
Request Parameters:
    {general-params}
    "interest_list":
    {
```

```
        "recommend_group":11,
        "select_duration":91,
        "interest_list": {
            [{"id":"Fitness & Health"},
            {"id":"OddlySatisfying"},
            {"id":"Travel"},
            {"id":"Comedy"}]
        }
    }

Response Body (protobuf):
    [
    ...
        {
            1: {tiktok-id}
            3: {author name}
            4: {caption}
            ...
            1: https://p16-sign-sg.tiktokcdn.com/...webp?... // link for image
            2: https://p16-sign-sg.tiktokcdn.com/...jpeg?... // link fore image
            ...
        }
    ...
    ]
```

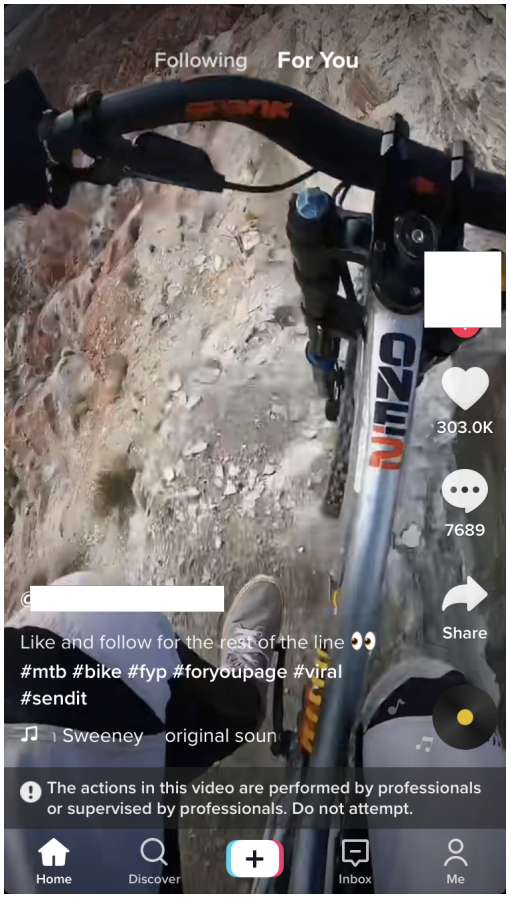


Figure 4.6: *TikTok's* For You page (FYP).

The user's previously indicated interests are provided to this endpoint. As the app is used more, the user is assigned *'recommend_groups'* as shown in the above request body. While there is no direct evidence for what the nature of these groups is, it is deducible that these groups resemble topics of interest, which the user is assigned to based on how they engage with different content. The response to the above endpoint contains information about the videos to be displayed to the user, like their creators, URLs for the videos themselves, video labels, etc. The video and image content (images for creator profile picture) is retrieved from a server cluster called *tiktokcdn.com*.

```
GET https://p16-amd-va.tiktokcdn.com/img/musically-maliva-obj/16..70~noop.webp
    Response Type: image/jpeg

GET https://v19.tiktokcdn.com/0c3...29c/607713df/video/tos/useast2a/....
    Response Type: video/mp4
```

The main way in which the user can interact a video is by 'liking' it to show their approval or enjoyment of the content. A piece of *TikTok* content is identified by an *'aweme_id'*.

```
GET https://api16-normal-c-useast1a.tiktokv.com/aweme/v1/commit/item/digg/
```

```
Request Body:
    aweme_id:   6948450097461988614 // content ID
    channel_id: 0
    enter_from: homepage_hot
```

The user can hold the video and choose the 'Not interested' option to express dislike for the video, which spawns the same connection as above except to an endpoint called */dislike*. This sends the creator's *author_id* which is assumably used by the back-end to avoid displaying any other content by this creator to the user.

```
GET https://api16-normal-c-useast1a.tiktokv.com/aweme/v1/dislike/item/
Request Parameters:
    {general-params}
    aweme_id:   6948450097461988614 // content ID
Request Body:
    author_id:  6942449334701376518 // author ID
    channel_id: 0
    video_type: 0
```

Other ways the user can interact with content include commenting on the video or sharing it, both of which are carried out by the following endpoints respectively.

```
POST https://api-va.tiktokv.com/aweme/v1/comment/publish/
Request Parameters: {general-params}
Request Body:
    aweme_id: 6948100149482687749
    ...
    text: "Test" // comment text

GET https:api-va.tiktokv.com/shorten/
Request Parameters:
    {general-params}
    aweme_id: 6948100149482687749
Response Body:
    "data": "https://vm.tiktok.com/ZMeu7PyfY/" // sharable link
```

All of these actions signal to *TikTok's* back-end how the user engages with different content. It is obvious from the nature of the application that the back-end processes this information and builds a profile of what topics the user is interested in, and serves content based on this. While unconfirmed, it is not unreasonable to speculate this is carried out by assigning the user to groups as previously mentioned. These groups are pulled by the user's device from the */notice/count* endpoint.

```
GET https://api-va.tiktokv.com/aweme/v1/notice/count...
```

```
Request Parameters: {general-params}
Response Body:
    "groups_in_filters": [
        {
            "filter_type": 3,
            "groups": [
                6,
                14
            ]
        },
        ...
        {
            "filter_type": 6,
            "groups": [
                84
            ]
        },
        ...
        {
            "count": 0,
            "group": 86,
            "show_type": 2
        },
        ...
        {
            "clear_occasion": 1,
            "count": 0,
            "group": 88,
            "show_type": 2
        },
```

If the assumption here is correct, the response here would inform the application on what groups or topics the user has interest in, perhaps designating their display priority by the *'filter_type'* or *'show_type'* values. In this way, the app builds a strong profile of the user's interests, which is an extremely powerful basis for targeted advertising.

With regard to monitoring and event logging, two connections occur very regularly - roughly every 20 seconds - to different servers. The first of which is an encrypted connection, which occurs to *log-va.tiktokv.com*. In order to decrypt the body of this request, a *Frida* script called *observeMethod.js* (**Appendix B**) was injected, targeting the method responsible for building requests to this endpoint, and print out its the methods arguments and return value. This script was adapted from another script

found at (66). The targeted method is called *'actualSendTrack:...:postParams:...'* and belongs to the *TTTrackerProxy* class. It was targeted based on its name likely indicating the sending of data. The results produced by this technique were verified to be correct by cross referencing the output with the findings of a blog post (67) which aimed to decrypt the same request spawning from an Android device. The full decoded request body is displayed in **Appendix E**, however a subsection of the more interesting data is shown below.

```
POST https://log-va.tiktokv.com/service/2/app_log/...
Request Parameters:
    {general-params}
Request Body:
    header =     {
        ...
        custom = {
            ...
            "earphone_status" = off;
            ...
        };
        ...
        "is_jailbroken" = 1;
    };
    "event_v3" = (
        {
            ...
            datetime = "2021-04-15 12:05:26";
            event = "ug_skadnet_conversion_value";
            data = {
                "conversion_value" = 47;
                "first_activation_time" = 1618394913;
                "launch_duration" = 25;
                "session_launch" = 2;
                "sum_duration" = 2437;
                vv = 0;
            };
        },
        {
            datetime = "2021-04-15 12:05:51";
            event = "ug_skadnet_conversion_value";
            data = {
                "conversion_value" = 47;
```

```
            "first_activation_time" = 1618394913;
            "launch_duration" = 25;
            "session_launch" = 2;
            "sum_duration" = 2462;
            vv = 0;
        };
    }
);
```

While one might expect more sensitive data to be transmitted in this encrypted connection, the request body does not contain anything immediately concerning. Some interesting observations might be parameters such as *'earphone_status'* to designate if the user is using an external audio device, or perhaps some of the data included in the *'event_v3'* object. This contains data about certain events that occur within the app, and information about how long they take (*'sum_duration'*) and when they first occurred (*'first_activation_time'*). As mentioned, this connection is extremely frequent and was verified to transmit data about a variety of different events. Some examples of event names observed include *'ug_skadnet_conversion_value'*, *'PUSH_SDK_synchronize'*, and *'stay_time'* to name but a few. This accounts for the contents of the first of the previously listed encrypted connections. The second, involves communications with a *'/monitor'* endpoint. If the class - method *'HMDHeimdallrConfig performanceReportURL* is observed, the return string is one of interest.

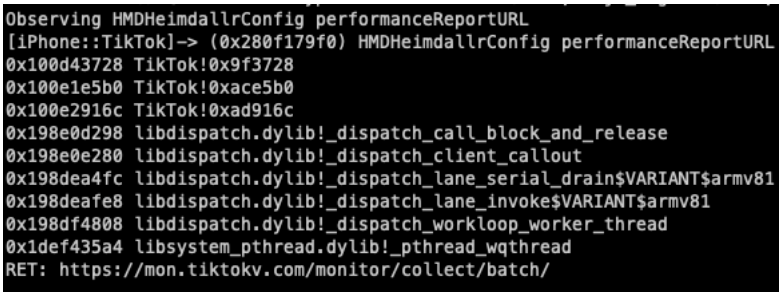


Figure 4.7: Interception of class-method *HMDHeimdallrConfig performanceReportURL*

While this might indicate the class responsible for building this request, more information could not be discerned about this connection, meaning its contents could not be decrypted. While one might speculate due to the method name above that this connection regards performance monitoring, it would not be reasonable to speculate that this is the only kind of data being transmitted in the request. Ideally, more work could be done to uncover the nature of this connection, and others to the *mon.tiktokv.com* server.

d. Third Parties

While *TikTok* draws briefly on services from Facebook and Snapchat for sign-in flows and camera filters respectively, *Firebase* and *AppsFlyer* have connections occurring more regularly as the app is used. In both of these cases, a report is sent to both services when the app is closed after a session of use. Looking at communications with *Firebase*, the app behaves in similar fashion to *Aarogya Setu*, whereby error reports and usage information is stored in the Google service. Nothing new or hugely interesting is transmitted in these connections. In the case of *AppsFlyer*, this is a company that caters advertising attribution and marketing analytics. There are multiple connections that occur to this service, however the request body of each is encrypted. In order to discover the contents, the following strategy was implemented.

1. Use *frida-trace* to search for classes including the substring 'AppsFlyer'
2. Observe the '*executeRequestWithUrlString*' function from the *'AppsFlyerLib'* class being called.
3. Intercept the arguments and return value of this function using *observeMethod.js* (**Appendix B**)
4. This method returns a url, *https://launches.appsflyer.com/...*, which is one of the connections we want to monitor.
5. The parameters of this request are created by another method, called *dict* from the *AppsFlyerDictionary* class. Target this function with *observeMethod.js*, and what is likely the request contents of this connection can be observed.

```
POST https://launches.appsflyer.com/api/v6.1/iosevent?app_id=835599320&buildnumber=6.1.4
Request Body:
    {
        JBDevice = 1;
        advertiserId = "7F7...059";
        "af_timestamp" = 1618594033824;
        counter = 79;
        ...
        "dev_key" = XY8Lpakui8g4kBcposRgxA;
        deviceData =     {
            "cpu_64bits" = true;
            "cpu_count" = 6;
            "cpu_speed" = "-1";
            "cpu_type" = "ARM64_V8";
            "device_model" = "iPhone10,1";
            dim =         {
```

```
            "x_px" = 750;
            "y_px" = 1334;
        };
        osVersion = "14.1 (Build 18A8395)";
        "ram_size" = 1974;
    };
    disk = "42893/61005";
    event = Launched;
    eventName = Launched;
    firstLaunchDate = "2021-04-15_152604+0100";
    ...
    reinstallCounter = 7;
    "sc_o" = p;
    sessioncounter = 11;
    timepassedsincelastlaunch = 60;
    timestamp = "1618594033.824621";
    wifi = 1;
}
```

Device information like whether it is jailbroken, the amount of times it has been reinstalled, and CPU information are among the data being collected. This occurs whenever the user closes the app or locks the screen on the app, serving as a regular update on this information for the *Apps Flyer* service. Another two connections to this service were observed, however they could not be decrypted or discovered through use of *Frida* within the time-frame of this work. Those connections occur less frequently, perhaps once every instance of the application.

- *https://conversions.appsflyer.com/*
- *https://inapps.appsflyer.com/*

Due to its similarity of *Apps Flyer's* service to that of *mParticle's* - as is evident in *Airbnb* - it would be reasonable to assume the data being transmitted here is of a similar nature, although clearly this could not be verified.

e. Observations

Having garnered enough attention to be considered for a ban by the American government might cause one to believe *TikTok* is performing some sort of undesirable activity with regards to security and privacy of its users. However, from what this work has discovered, nothing outside the ordinary is being transmitted. Of course, the concept of token linking is certainly present, with common parameters being used constantly in lots of connections, and usage of a lot of different IDs (*cdid, device_ id,*

openudid). This constant transmission of identifiers is surely avoidable, and would be suitably replaced by a single identifier for the user or their device. The parameter identifiers *cdid, vid* and *openudid* were found to persist for the duration of an the app's installation, along with the *device_id* value. All of these however, were verified to change when the app was deleted an reinstalled. The app also uses cookies in a lot of its back-end requests. This seems unnecessary because all requests are linked by the IDs being transmitted anyway, from which the app's back-end could use to identify a user across sessions, eliminating the need for cookies. While the app uses a third party library for certificate pinning, the mere fact that all connections are unpinned and therefore insecure on the first instance of the app is a noteworthy observation about the app's security in its own right. Surely a more secure strategy could be employed here.

Another striking observation while examining *TikTok's* traffic is the volume of requests, two of which specifically occur every 20 seconds. The first of these is a POST request to the *'/service/2/app_log/'* which sends event logging information to the server. The other of these connections is a GET method request to the */skan* endpoint, which essentially keeps the application updated on how long it has been running, and the number of sessions it has experienced. The frequency of these requests could surely be avoided, with updates perhaps really only needed when the app is opened or closed.

While the exact details of the connections to *AppsFlyer's* service could not be discerned, it is assumable that its purpose is similar in nature to the previously discussed *mParticle* service, and how that is used in *Airbnb*. This would mean information about how the user interacts with certain content is made available to *AppsFlyer*, who could send it on to social media partners for advertising purpose. An important point here, is that when the user disables targeted advertising in the iPhone's settings, the *idfa* value is nullified in these requests, which limits *AppsFlyer's* ability to track the user for advertising purposes.

As a final point, it is important to discuss the nature of *TikTok's* core functionality, and the strategy they employ to make their app addictive to users. As a user interacts with *TikTok* over the course of time the application its back-end services build a very strong profile of their user's interests, beliefs and perhaps even political leanings so they can serve relevant content. This is an incredibly powerful basis for advertising, and when highlighted might encourage users to think about how their behaviour and interest patterns are being monitored. The popular phrase used to describe advertising, 'If you are not paying for the product, you are the product', might be used to aptly sum-up the nature of these interactions. With this in mind, users might be encouraged to think about how much an app like *TikTok* can deduce about them, which might not only

include interests, but also political leanings, religious beliefs and other patterns of behaviour that can reveal a lot about a user. It is certainly worth considering and informing users before using an app like *TikTok*, if they are happy to surrender this aspect of their privacy.

CHAPTER 5

Conclusion

5.1 Future Work

This work outlines a formal basis by which any iOS application can be deconstructed and analysed through a lens of user privacy. Of course different applications employ different security measures, such as encryption, and also draw on different third party services. The general methodology defined in this work however, could be applied to any iOS application, on iOS for version 14.1.

In one or two cases specifically regarding *TikTok*, the contents of some internet traffic could not be uncovered within the time frame of this work. Such a popular application certainly merits further exploration into exactly what these connections are encrypting and sending. Furthermore, applications are constantly changing and updating their implementations and usage of different services. It is important that these updates are always geared towards increasing security of maintaining the privacy of the app's users.

5.2 Conclusion

For the most part, the data of iPhone users is secured and transmitted in accordance with the best policies for protection of user's privacy. There are however cases, which have been outlined at length above, in which transmission of data is questionable, handled poorly or seemingly unnecessary altogether. Specifically regarding Apple, frequent transmissions of sensitive identifiers are observed, enabling the linking user's

requests across various different Apple services. Some questionable transmissions of data also do occur, such as that of local MAC addresses connected to the same network as the iPhone. Apple do also however provide the basis for users to limit a lot of the traffic spawning from an iPhone, as discussed in section 4.1.g. This does allow users to retain control over their information, but largely defeats the point of a smartphone.

Sometimes applications acquire data that they do not appear to need, or if required is excessively accurate. For the provision of a travel lodging application like *Airbnb*, information about the user's device magnetic readings or link-local IP addresses do not immediately appear to be important for the functioning of the app, and collection of this information would ideally be avoided. Further demonstrations of this can be observed in *Aarogya Setu*, who acquire GPS coordinates when it really could be substituted for a vague representation of location. *TikTok* also sends a huge chunk of parameters which identify the user in the vast majority of requests it makes to its servers, a lot of which appear to be entirely avoidable. Vendors should ideally aim to collect as little data as possible for the functioning of the app. In the case personal information is required about the user, efforts should be made to see if obfuscating that data is a viable course of action, such as that witnessed in *COVID Tracker Ireland*.

While Apple has control over what apps are made available on the App Store, it is largely up to application vendors to define a privacy policy and protect their user's data in a manner that adheres with best practices. It has been observed at many points throughout the course of this work that at times, app vendors take liberties with their user's data, and are often not hugely transparent about how they use this data, which is particularly evident in *Airbnb* & to a degree, *TikTok*. In other cases, such as that of *Aarogya Setu*, the user can often be unaware - unless they read the app's privacy policy in detail - about what data is being uploaded to servers and at what times. This highlights an important point about privacy policies, in that they are used by vendors to state their intentions and compliance with user privacy laws, in the knowledge that only a tiny fraction of users will actually read them before consenting to their terms. It is clear more could be done in ensuring users are made aware of how their data is being used in a more transparent manner. A slightly more difficult case is that of *TikTok*, whereby collection of data about how users interact with different types of content isn't really eliciting sensitive information, however over time allows the amassing of data with potential to build a very accurate profile of a user. This is harder to define in a privacy policy, and undoubtedly goes unconsidered by a lot of *TikTok's* users.

While the use of a third party service like Google's *Firebase* for storage is entirely reasonable and in some cases a good option for the storage of application data, ideally usage of third party services would be kept to a minimum. In the event they are used, little sensitive information should be stored on third party servers. In the case of

Airbnb, several third parties are used and in regular communication with the app. A lot of these services are also sent identifiers like the *x-airbnb-device-id* and *IDFA*, which provides them with an identifier for the user. This of course is not ideal.

At multiple times, advertising has become a relevant topic of discussion during this work. Somewhat unfortunately, advertising has become one of the dominant avenues by which app vendors make profit, which essentially means they are monetising their users data and patterns of behaviour. Companies like Apple are increasing efforts in allowing users to control and limit how their data is used for targeted advertising, however they can only do so much, as applications turn to third party advertising services like *mParticle*. Apps like *TikTok* build such a strong profile of their users that one can imagine how shocking it would be for a user to see how much the app knows about them. The power that users bestow upon companies that own these applications is formidable, and should be treated with much more care. With all of these findings in mind, it is important to consider the question as to if the technology companies in control of all this data are doing enough to keep it safe and secure. Events such as data breaches in Facebook (2019), which leaked the account data of approximately 500 million users (68) certainly do not inspire confidence. As a concluding statement for this work, it becomes evident through analysis of an iPhone's interaction with the internet that the state of user privacy in the world today still has a long way to go, before the full security and privacy of internet users can be assured. There are however, reasons to be hopeful for the future.

Bibliography

[1] List of trusted root certificates in iOS 14.
`https://support.apple.com/en-gb/HT212140`

[2] *SSL Killswitch 2* - An open source blackbox tool for disabling certificate pinning validation on iOS, created by *nabla-c0d3*
`https://github.com/nabla-c0d3/ssl-kill-switch2`

[3] *mitmproxy* - A hacking tool for intercepting traffic.
`https://mitmproxy.org/`

[4] *mitmproxy* transparent mode setup.
`https://docs.mitmproxy.org/stable/concepts-modes/#transparent-proxy`

[5] Eric Smith, (2010) *iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs)*
`http://pskl.us/wp/wp-content/uploads/2010/09/iPhone-Applications-Privacy-Issues.pdf`

[6] Checkra1n jailbreaking software tool.
`https://checkra.in/`

[7] iPhone wiki for *Cydia Substrate*.
`https://iphonedevwiki.net/index.php/Cydia_Substrate`

[8] *Frida* - A dynamic instrumentation toolkit used for reverse engineering.
`https://frida.re/`

[9] Article discussing the how *Aarogya Setu* was made mandatory in May 2020.
`https://indianexpress.com/article/technology/social/aarogya-setu-app-mandatory-contact-tracing-app-6389284/`

[10] Statistics on worldwide iPhone usage.
https://www.statista.com/statistics/755625/iphones-in-use-in-us-china-and-rest-of-the-world/#:~:text=In%20April%202017%2C%20728%20million,120%20million%20in%20the%20U.S

[11] React library *react-native-ios11-devicecheck* - Used to check iOS device information.
https://github.com/dayitv89/react-native-ios11-devicecheck

[12] *COVID Tracker Ireland* repository source code (React).
https://github.com/HSEIreland/covid-tracker-app

[13] Apple Developer support page for the DCDevice class.
https://developer.apple.com/documentation/devicecheck/dcdevice/2902276-generatetokenwithcompletionhandl?language=objc

[14] Source code showing how *react-native-ios11-devicecheck* generates the *deviceVerificationPayload*.
https://github.com/dayitv89/react-native-ios11-devicecheck/blob/e037405c4aa5a304c031ed9a32a9d9d14afa5420/ios/RNIOS11DeviceCheck.m#L30

[15] Source code showing how *COVID Tracker Ireland* sends the *deviceVerificationPayload* onward to Apple servers.
https://github.com/HSEIreland/covid-tracker-backend-api/blob/851409d46c66749a2228206678af9b1b83680127/lib/plugins/verify/index.js#L74

[16] RFC 719 outlining JSON Web Tokens (JWT).
https://tools.ietf.org/html/rfc7519

[17] *jwt.io* - JWT decoder tool.
https://jwt.io/

[18] Google *Firebase* service.
https://firebase.google.com/

[19] Protocol buffers.
https://developers.google.com/protocol-buffers

[20] Temporary Indian phone number site.
http://receive-sms-online.info/919532437525-India

[21] *Aarogya Setu* source code; Occurrence of Bluetooth uploads.
https://openforge.gov.in/plugins/git/aarogyasetuos/ios-mobile-application?a=

blob&hb=f5da99c25a9aa7f2b0a1f18ef841c14a83593a3d&h=
703f652aea492aad622dfc645c8045ee0ff8fc8f&f=CoMap-19%2FAppDelegate.
swift#L322

[22] *mitmproxy* interceptions.
https://docs.mitmproxy.org/stable/mitmproxytutorial-interceptrequests/

[23] *Frida's* core of instrumentation - Gum.
https://github.com/frida/frida-gum

[24] *Frida's* documentation.
https://frida.re/docs/modes/

[25] "The engineering behind reverse engineering" - a presentation about how *Frida* works.
https://frida.re/slides/
osdc-2015-the-engineering-behind-the-reverse-engineering.pdf

[26] *frida-trace* - command for tracing remote processes functions.
https://frida.re/docs/frida-trace/

[27] *mmap* linux command.
https://man7.org/linux/man-pages/man2/mmap.2.html

[28] *dlopen* linux command.
https://man7.org/linux/man-pages/man3/dlopen.3.html

[29] Cydia
https://cydia-app.com/

[30] Srivatsa & Hicks, (2010), *Deanonymizing mobility traces: Using social networks as a side-channel*
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.365.5128&rep=
rep1&type=pdf

[31] *COVID Tracker Ireland* privacy policy
https://github.com/HSEIreland/covidtracker-documentation/blob/master/
documentation/privacy/DPIN%20IE%20EFGS%20Updated%20FINAL%2015.10.2020.
pdf

[32] Google *Firebase* documentation on client identifiers.
https:
//firebase.google.com/docs/projects/manage-installations#objective-c_3

[33] *Aarogya Setu* iOS source code.
https:
//openforge.gov.in/plugins/git/aarogyasetuos/ios-mobile-application

[34] *Aarogya Setu* Privacy Policy.
https://web.swaraksha.gov.in/ncv19/privacy/

[35] Singapore's Contact Tracing app: *TraceTogether*.
https://www.tracetogether.gov.sg/

[36] Apple calendar configuration developer documentation *Caldav*.
https://developer.apple.com/documentation/devicemanagement/caldav

[37] Apple Push Notification Service documentation (APNs).
https://developer.apple.com/library/archive/documentation/
NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#
//apple_ref/doc/uid/TP40008194-CH8-SW1

[38] Apple's documentation & information on advertising and privacy.
https://support.apple.com/en-us/HT205223

[39] A. Pultier, N. Harrand & P.B. Brandtzaeg, (2016), *Privacy in mobile apps*. Available here.

[40] Apple documentation on the CLGeocode Obj-C class.
https://developer.apple.com/documentation/corelocation/clgeocoder?
language=objc

[41] Cuebiq
https://www.cuebiq.com/

[42] GDPR laws and regulations.
https://gdpr.eu/

[43] N. Seriot, (2010), *iPhone Privacy*
http://seriot.ch/resources/talks_papers/iPhonePrivacy.pdf

[44] M. Egele et. al. (2011), *PiOS: Detecting Privacy Leaks in iOS Applications*.
http://www.syssec-project.eu/m/page-media/3/egele-ndss11.pdf

[45] *COVIDSafe* - Australia's COVID tracking app.
https://www.health.gov.au/resources/apps-and-tools/covidsafe-app

[46] K. Michael & R. Abbas, (2020) *Behind COVID-19 Contact Trace Apps: The Google-Apple partnership*.
https://ieeexplore.ieee.org/abstract/document/9117186

[47] H. Cho, D. Ippolito & Y. W. Yu, (2020), *Contact Tracing Mobile apps for COVID-19: Privacy considerations and related trade-offs.*
https://arxiv.org/pdf/2003.11511.pdf

[48] T. Sharma & M. Bashir, (2020), *Use of apps in the COVID-19 response and the loss of privacy protection.*
https://www.nature.com/articles/s41591-020-0928-y?mc_cid=79a90b1fb5&mc_eid=547d4476efs

[49] D. Leith & S. Farrell, (2020), *UseContact Tracing App Privacy: What Data Is Shared By Europe's GAEN Contact Tracing Apps*
https://www.scss.tcd.ie/Doug.Leith/pubs/contact_tracing_app_traffic.pdf

[50] HSE Data Protection Impact Assessment (DPIA) of *COVID Tracker* Ireland.
https://github.com/HSEIreland/covidtracker-documentation/blob/master/documentation/privacy/IE%20Data%20Protection%20Impact%20Assessment%20for%20the%20COVID%20Tracker%20App%20EFGS%20update%20FINAL%2015.10.2020.pdf

[51] P. Dehaye & J. Reardon (2020) *Proximity Tracing in an Ecosystem of Surveillance Capitalism*
https://arxiv.org/pdf/2009.06077.pdf

[52] Find my iPhone service.
https://www.apple.com/icloud/find-my/

[53] *frida-ios-dump - AloneMonkey's* script for dumping an iOS app file.
https://github.com/AloneMonkey/frida-ios-dump/blob/master/dump.py

[54] Google's OAuth protocol.
https://developers.google.com/identity/protocols/oauth2

[55] Google's reCaptcha service.
https://www.google.com/recaptcha/about/

[56] RFC4291 - IPv6 Link local addresses.
https://tools.ietf.org/html/rfc4291

[57] C. Tejada, J. Carrasco-Jimenez & R. Brena, (2013), *Location Identification Using a Magnetic-field-based FFT Signature.*
https://www.researchgate.net/publication/242061764_Location_Identification_Using_a_Magnetic-field-based_FFT_Signature

[58] Business Insider article on Donald Trump's attempted *TikTok* ban.
https://www.businessinsider.com/donald-trump-tiktok-ban-us-china-explained-in-30-seconds-2020-8?r=US&IR=T

[59] OneTrust.
https://www.onetrust.com/

[60] CookiePro article about the usage of geolocation service.
https://community.cookiepro.com/s/article/UUID-fe9e37a5-1560-433e-abcb-642ce2073ad4

[61] mParticle website.
https://www.mparticle.com/

[62] mParticle information about parters.
https://www.mparticle.com/integrations/category/advertising

[63] Airbnb's privacy policy.
https://www.airbnb.ie/help/article/2855/privacy-policy?locale=en&_set_bev_on_new_domain=1617360138_Njk3YTAzYzA1MTdm

[64] Snapkit.
https://snapkit.com/

[65] AppsFlyer
https://www.appsflyer.com/

[66] Frida script for printing function arguments and return value.
https://codeshare.frida.re/@mrmacete/objc-method-observer/

[67] Blog post on decrypting log connections from *TikTok* on Android.
https://medium.com/@fs0c131y/tiktok-logs-logs-logs-e93e8162647a

[68] Business Insider article on Facebook data leak.
https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4?r=US&IR=T

[69] Apple & Google's Exposure Notification Service (ENS)
https://www.google.com/covid19/exposurenotifications/

[70] Article describing the mandatory nature of India's COVID tracker app.
https://www.reuters.com/article/us-health-coronavirus-india-app-idUSKBN22E07K

[71] Bugsnag
https://www.bugsnag.com/

Appendices

Appendix A - Decode a protobuf - *decode_protobuf.py*

```
from mitmproxy import http
import zlib
import subprocess
from codecs import encode, decode
import os
import json
import codecs

################################################
# Run with:
# mitmdump -nr <an mitm flow> -s decode_protobuf.py
################################################

class Decode:
    def __init__(self):
        print("- Initialise -")

    def decode_pb(self, bites):
        f = open('dp', 'wb')
        f.write(bites)
        f.close()

        try:
            print("> Decoded protobuf")
            os.system("cat dp")
        except:
            return "Failed"

    def response(self, flow: http.HTTPFlow):
        focus_url = "https://fp.swaraksha.gov.in/api/v3/users/data/"

        if flow.request.pretty_url == focus_url:
            print("=======================================")
            try:
                print("Attempt 1: Protobuf, Zlib decode")
                pb = self.decode_pb(
                    zlib.decompress(
                        flow.request.raw_content,
                        32 + zlib.MAX_WBITS, )
                    )
            except:
                print("Attempt 2: Try simple decode as utf8")
                pb = self.decode_pb(flow.request.raw_content)
                try:
```

```
                    pb = (flow.request.raw_content.decode('utf-8') )
                    print("2: ", pb)
                except:
                    print("Attempt 3: Bytes --> ascii")
                    pb = bytes(flow.request.raw_content)

                    for i in range(len(pb)):
                        print(chr(pb[i]), end="")
            print("\n=======================================\n\n")

addons = [Decode()]

def hexdump(src, length=16):
    FILTER = ''.join([(len(repr(chr(x))) == 3) and chr(x) or '.'
        for x in range(256)])
    lines = []
    for c in range(0, len(src), length):
        chars = src[c:c+length]
        hex = ' '.join(["%02x" % ord(x) for x in chars])
        printable = ''.join(["%s" % ((ord(x) <= 127 and FILTER[ord(x)])
            or '.') for x in chars])
        lines.append("%04x  %-*s  %s\n" % (c, length*3, hex, printable)
            )
    return ''.join(lines)
```

Appendix B - Observe class method *observeMethod.js*

```
// SCRIPT adapted from the script found here: https://codeshare.frida.re/@mrmacete/objc-method-observer/
// - Adapted to target TTTrackerProxy which builds an encrypted request for TikTok
// - Intercepting this method allows examination of parameters before encryption.

const classN = "TTTrackerProxy";
const methodName = "actualSendTrack:trackerIDs:v3TrackIDs:URLString:headerField:postParams:needEncrypt:policy:";

var ISA_MASK = ptr('0x0000000ffffffff8');
var ISA_MAGIC_MASK = ptr('0x000003f000000001');
var ISA_MAGIC_VALUE = ptr('0x000001a000000001');

const c = ObjC.classes[classN][`- ${methodName}`];
console.log(`Intercepting: ${c}`);
```

```
14
15 observeMethod(c.implementation, classN, methodName);
16
17 function observeMethod(impl, name, m) {
18     console.log('Observing ' + name + ' ' + m);
19     Interceptor.attach(impl, {
20         onEnter: function(a) {
21             this.log = [];
22             this.log.push('(' + a[0] + ') ' + name + ' ' + m);
23             if (m.indexOf(':') !== -1) {
24                 var params = m.split(':');
25                 params[0] = params[0].split(' ')[1];
26                 for (var i = 0; i < params.length - 1; i++) {
27                     if (isObjC(a[2 + i])) {
28                         const theObj = new ObjC.Object(a[2 + i]);
29                         this.log.push(params[i] + ': ' + theObj.
                               toString() + ' (' + theObj.$className + ')'
                               ;
30                     } else {
31                         this.log.push(params[i] + ': ' + a[2 + i].
                               toString());
32                     }
33                 }
34             }
35             this.log.push(Thread.backtrace(this.context, Backtracer.
                   ACCURATE)
36                 .map(DebugSymbol.fromAddress).join("\n"));
37         },
38         onLeave: function(r) {
39             if (isObjC(r)) {
40                 this.log.push('RET: ' + new ObjC.Object(r).toString());
41             } else {
42                 this.log.push('RET: ' + r.toString());
43             }
44
45             console.log(this.log.join('\n') + '\n');
46         }
47     });
48 }
49
50 function isObjC(p) {
51     var klass = getObjCClassPtr(p);
52     return !klass.isNull();
53 }
54
55 function getObjCClassPtr(p) {
56     if (!isReadable(p)) {
```

```
57          return NULL;
58      }
59      var isa = p.readPointer();
60      var classP = isa;
61      if (classP.and(ISA_MAGIC_MASK).equals(ISA_MAGIC_VALUE)) {
62          classP = isa.and(ISA_MASK);
63      }
64      if (isReadable(classP)) {
65          return classP;
66      }
67      return NULL;
68  }
69
70  function isReadable(p) {
71      try {
72          p.readU8();
73          return true;
74      } catch (e) {
75          return false;
76      }
77  }
```

Appendix C - Observe all methods of class *observeClass.js*

```
1  // Script taken from https://codeshare.frida.re/@mrmacete/objc-method-
       observer/
2  // Used to view ALL methods of a specified class
3  // Allows examination of arguments and return values of every method in
       the class.
4
5  var ISA_MASK = ptr('0x0000000ffffffff8');
6  var ISA_MAGIC_MASK = ptr('0x000003f000000001');
7  var ISA_MAGIC_VALUE = ptr('0x000001a000000001');
8
9  function observeSomething(pattern) {
10     var resolver = new ApiResolver('objc');
11     var things = resolver.enumerateMatchesSync(pattern);
12     things.forEach(function(thing) {
13         observeMethod(thing.address, '', thing.name);
14     });
15 }
16
17 function observeClass(name) {
18     var k = ObjC.classes[name];
```

```
19      if (!k) {
20          return;
21      }
22      k.$ownMethods.forEach(function(m) {
23          observeMethod(k[m].implementation, name, m);
24      });
25  }
26
27  function observeMethod(impl, name, m) {
28      console.log('Observing ' + name + ' ' + m);
29      Interceptor.attach(impl, {
30          onEnter: function(a) {
31              this.log = [];
32              this.log.push('(' + a[0] + ') ' + name + ' ' + m);
33              if (m.indexOf(':') !== -1) {
34                  var params = m.split(':');
35                  params[0] = params[0].split(' ')[1];
36                  for (var i = 0; i < params.length - 1; i++) {
37                      if (isObjC(a[2 + i])) {
38                          const theObj = new ObjC.Object(a[2 + i]);
39                          this.log.push(params[i] + ': ' + theObj.
                                toString() + ' (' + theObj.$className + ')')
                                ;
40                          this.log.push(`Stink worth: ${(theObj.handle.
                                readUtf8String())}`);
41                      } else {
42                          this.log.push(params[i] + ': ' + a[2 + i].
                                toString());
43                      }
44                  }
45              }
46
47              this.log.push(Thread.backtrace(this.context, Backtracer.
                    ACCURATE)
48                  .map(DebugSymbol.fromAddress).join("\n"));
49          },
50
51          onLeave: function(r) {
52              if (isObjC(r)) {
53                  this.log.push('RET: ' + new ObjC.Object(r).toString());
54                  console.log(`LEAVE: ${r.toString()}`);
55              } else {
56                  console.log(`LEAVE: ${r}`);
57                  this.log.push('RET: ' + r.toString());
58              }
59
60              console.log(this.log.join('\n') + '\n');
```

```
        }
    });
}

function isObjC(p) {
    var klass = getObjCClassPtr(p);
    return !klass.isNull();
}

function getObjCClassPtr(p) {
    if (!isReadable(p)) {
        return NULL;
    }
    var isa = p.readPointer();
    var classP = isa;
    if (classP.and(ISA_MAGIC_MASK).equals(ISA_MAGIC_VALUE)) {
        classP = isa.and(ISA_MASK);
    }
    if (isReadable(classP)) {
        return classP;
    }
    return NULL;
}

function isReadable(p) {
    try {
        p.readU8();
        return true;
    } catch (e) {
        return false;
    }
}

observeClass("AFSDKiAdClient"); // enter the class you want to monitor
   here.
```

Appendix D - Deregistration connections for iCloud

```
DELETE https://pr-pod3-smp-device.apple.com/broker/v4/device/04...67
Request Headers:
            X-apple-i-md-d: GH...6o2
            X-apple-i-md: AAA..w==

POST https://gateway.icloud.com/ckdevice/api/client/pushUnregister
Request Headers:
```

```
            X-apple-i-md-d: GH...6o2
            X-apple-i-md: AAA..w==
Request Body:
            54b...95a (uuid)
            F4G...6H (serial number)

POST https://setup.icloud.com/setup/account/deregisterDevice
Request Headers:
            X-apple-i-md-d: GH...6o2
            X-apple-i-md: AAA..w==
Request Body:
       <key>appleId</key>
      <string>tkelly2@tcd.ie</string>
      <key>backupDeviceUUID</key>
      <string>D:e91f.....39a5</string>
      <key>dsid</key>
      <string>173....6</string>
      <key>pushToken</key>
      <string>403....A48</string>
      <key>serialNumber</key>
      <string>F4G....6H</string>
      <key>udid</key>
      <string>54b....95ba</string>

POST https://profile.gc.apple.com/WebObjects/GKProfileService.woa/wa/logoutUser
Request Headers:
            X-apple-i-md-d: GH...6o2
            X-apple-i-md: AAA..w==

POST https://buy.itunes.apple.com/WebObjects/MZFinance.woa/wa/logout
Request Headers:
            X-apple-i-md-d: GH...6o2
            X-apple-i-md: AAA..w==

Request Body:
            54b5....95ba (uuid)
```

Appendix E - Request sent to */app_log* endpoint

```
POST https://log-va.tiktokv.com/service/2/app_log/...
Request Parameters: {general-params}
```

```
Request Body:
    header =     {
        access = WIFI;
        aid = 1233;
        ...
        cdid = "5FA...312";
        channel = "App Store";
        custom =         {
            "app_language" = en;
            "app_region" = IE;
            "build_number" = 190016;
            "earphone_status" = off;
            "filter_warn" = 0;
            "is_kids_mode" = 0;
            "user_mode" = 0;
            "user_period" = 0;
        };
        "device_id" = 695...213;
        idfa = "7F7...059";
        "install_id" = 695...110;
        "is_jailbroken" = 1;
        "os_version" = "14.1";
        package = "com.zhiliaoapp.musically";
        "vendor_id" = "AB0...415";
    };
    "event_v3" =     (
        {
            "ab_sdk_version" = "50042294,50053182,70100883,70105287,0,50017293";
            datetime = "2021-04-15 12:05:26";
            event = "ug_skadnet_conversion_value";
            nt = 4;
            params = {
                "error_code" = 0;
                "local_time_ms" = 1618484726425;
                nt = 4;
                "res_data" = {
                    "_AME_Header_RequestID" = 20210415110526010234100036 23EC4711;
                    code = 0;
                    data = {
                        "conversion_value" = 47;
                        "first_activation_time" = 1618394913;
```

```
                    "launch_duration" = 25;
                    "session_launch" = 2;
                    "sum_duration" = 2437;
                    vv = 0;
                };
                message = ok;
            };
            "tea_event_index" = 105;
        };
        "session_id" = "19D0483F-177E-4348-8E53-DF05B56AC094";
        },
        {
            "ab_sdk_version" = "50042294,50053182,70100883,70105287,0,50017293";
            datetime = "2021-04-15 12:05:51";
            event = "ug_skadnet_conversion_value";
            nt = 4;
            params =             {
            "error_code" = 0;
            "local_time_ms" = 1618484751929;
            nt = 4;
            "res_data" = {
                "_AME_Header_RequestID" = 20210415110551010234106029 39FCBB95;
                code = 0;
                data = {
                    "conversion_value" = 47;
                    "first_activation_time" = 1618394913;
                    "launch_duration" = 25;
                    "session_launch" = 2;
                    "sum_duration" = 2462;
                vv = 0;
                };
                message = ok;
        };
        "tea_event_index" = 106;
        };
        "session_id" = "19D0483F-177E-4348-8E53-DF05B56AC094";
    }
);
```