Privacy Concerns in Android Systems

Samuel Alberto Magalhães Fernandes
Mestrado em Segurança Informática
Departamento de Ciência de Computadores
2023

Orientador
Rolando da Silva Martins
Professor Auxiliar
Faculdade de Ciências da Universidade do Porto

Coorientador
João Miguel Maia Soares de Resende
Professor Auxiliar
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Professor Rolando Martins, for the opportunity to work on this project and for the profound learning experience it has represented in my academic journey.

I would also like to extend my thanks to my thesis co-supervisor, Professor João Resende, for the invaluable teachings, suggestions, and guidance that consistently steered the development of this dissertation work from inception to conclusion.

I wish to convey my appreciation to my fellow students for their motivation and the shared experiences that often served as the driving force propelling me forward.

Lastly, but by no means less significant, I want to offer my deepest thanks to my family and friends who patiently endured my absences, never faltering in their unwavering support, without which, undoubtedly, the successful completion of this project would not have been attainable.

Resumo

Os *smartphones* são uma presença ubíqua nas nossas vidas e constituem uma valiosa ajuda nas mais variadas tarefas quotidianas, tais como escolher um restaurante, encontrar uma farmácia ou programar uma viagem de lazer. Para oferecer esta vasta panóplia de serviços, estes dispositivos recolhem, de forma contínua, dados pessoais sobre o seu utilizador.

Embora o objetivo primário do acesso aos dados pessoais do utilizador seja justificado com o pressuposto de estes serem necessários para o normal funcionamento das aplicações, estes dados podem também ser utilizados para estabelecer perfis dos utilizadores, permitindo inferir as suas preferências de consumo, crenças religiosas, filiações políticas e até a sua orientação sexual.

Os dados pessoais dos utilizadores recolhidos pelas aplicações instaladas nos seus dispositivos podem, também, ser instrumentalizados no sentido de manipular o comportamento dos utilizadores, impactando a sua vida pessoal e, em alguns casos, o próprio funcionamento da sociedade.

Entre os dados do utilizador, recolhidos pelos *smartphones*, a localização encontra-se entre os mais sensíveis. Para além disso, estes dados são recolhidos e utilizados por um conjunto alargado de aplicações cujo funcionamento assenta em serviços baseados na localização do utilizador.

O acesso, por vezes em tempo real, à localização do utilizador representa um risco de privacidade e de segurança, sendo evidente uma crescente preocupação por parte dos utilizadores com o controlo do acesso, por parte de terceiros, a este tipo de dados sensíveis. Estas preocupações têm sido ampliadas com o aumento da cobertura mediática das questões em torno da violação da privacidade e da segurança dos utilizadores, no contexto da utilização de *smartphones*.

Considerando a posição hegemónica que os dispositivos Android têm no mercado de *smartphones*, propomos uma solução que procura oferecer aos utilizadores destes dispositivos a possibilidade de regular, casuisticamente, o acesso aos seus dados de localização pelas aplicações instaladas nos seus equipamentos.

A nossa solução permite que o utilizador possa optar por fornecer a sua verdadeira localização ou, em vez disso, uma localização fictícia. A localização fictícia, escolhida de forma casuística pelo utilizador, pode ser um ponto estático ou um trajeto, a pé ou de automóvel, ao longo de uma rota simulada, permitindo que as aplicações mantenham o seu normal funcionamento, sem comprometer a privacidade do utilizador.

Abstract

Smartphones are an ubiquitous presence in our lives and constitute valuable aids in various daily tasks, such as choosing a restaurant, finding a pharmacy, or planning a leisure trip. To offer this extensive range of services, these devices continuously collect personal data about the device user.

Although the primary objective of accessing the user's personal data is justified by the necessity for the normal operation of applications, these data can also be used to establish user profiles, allowing inferences about their consumption preferences, religious beliefs, political affiliations, and even their sexual orientation.

The personal data of users collected by applications installed on their devices can also be instrumentalized to manipulate user behaviour, impacting their personal lives and, in some cases, the functioning of society itself.

Location data is one of the most sensitive types of user data collected by smartphones, and it is utilized by a wide range of applications that rely on *location-based services* for their functionality.

Access, sometimes in real-time, to the user's location represents a risk to privacy and security. There is a growing concern among users about controlling third-party access to this type of sensitive data. These concerns have been magnified by increased media coverage of issues surrounding the violation of privacy and user security in the context of smartphone use.

Considering the dominant position that Android devices have in the smartphone market, we propose a solution that seeks to offer users of these devices the possibility to selectively regulate access to their location data by the applications installed on their equipment.

Our solution allows the user to choose to provide their actual location or, instead, a fictitious location. The fictitious location chosen randomly by the user can be a static point or a route, either on foot or by car, along a simulated path, allowing applications to maintain their normal functionality without compromising user privacy.

Contents

List of Tables

List of Figures

Acronyms

adb Android Debug Bridge 46, 47

AOT Ahead-of-time 11

API Application Programming Interface 8, 10, 12– 14, 16, 27, 29, 38– 40, 42, 49, 52, 54, 58– 60

APK Android Package 14, 16, 36, 43

ART Android Runtime 11, 12, 16, 49

DAC Discretionary Access Controls 16

DEX Dalvik Executable 42

ELF Executable and Linkable Format 11

GDPR General Data Protection Regulation 2

GMS Google Mobile Services 17

GPS Global Positioning System 1, 2, 33, 34, 41

GSG Google SafetyNet Attestation API 17

HAL Hardware Abstraction Layer 11, 12

HIV Human Immunodeficiency Virus 2

IL Intermediate Language 42

IP Internet Protocol 2, 70

J2ME Java 2 Platform, Micro Edition 19

JAR Java Archive 43

JIT Just-In-Time 11, 12

LBS Location-based Service 3, 5, 28, 34, 35, 63, 65

MAC Mandatory Access Controls 16

OS Operating System 3, 10, 16, 26, 45, 46

OTA Over-The-Air 17, 47, 48

PmP Protect my Privacy 30, 31

POI Points of Interest 5

ROM Read-Only Memory 22, 23, 36, 47

SDK Software Development Kit 12, 46

SMS Short Message Service 12

TWRP Team Win Recovery Project 47

VM Virtual Machine 43

VPN Virtual Private Network 70

YAHFA Yet Another Hooking Framework for Android 16, 49

Chapter 1

Introduction

The technological advancement we have witnessed in recent years has made it possible to create devices with high computing power, combined with their compact size, enabling anyone to carry them anywhere. This has allowed smartphones to become ubiquitous in the daily lives of the vast majority of people, regardless of their geographic location. It is estimated that the number of smartphone users worldwide will reach 7.49 billion by 2025 [1].

The increasing usage of smartphones has been driven by the emergence of a myriad of applications (also known as apps [1]) and business models, giving mobile devices [2] a central role in the lives of their users.

The possibility of having everything in a single device, coupled with the widespread availability of wireless networks anywhere (e.g., homes, hotels, gyms, shopping malls, restaurants, and universities), has led smartphone users to become dependent on these devices [2][3].

The evolution of wireless communication technology has allowed for a wide range of affordable data services to emerge. In parallel with this reality, all smartphones nowadays come equipped with Global Positioning System (GPS), which has been a determining factor in the emergence of a new generation of *location-based applications*, revolutionising the entire ecosystem surrounding the use of mobile devices. Alongside all the advantages, this transformation raises concerns about user privacy, particularly regarding the potential misuse of location data [4].

[1]Throughout this thesis, we use the terms apps and applications interchangeably.
[2]Throughout this thesis, we use the terms smartphones and mobile devices interchangeably.

The use of GPS-equipped smartphones enables users to directly share their location through various social media platforms such as *Facebook*, *Twitter*, and *Instagram*. In addition to this aspect, many of the applications installed on users devices require permission to access location data.

Access to location data on a large scale brings benefits to various stakeholders: the industry can create more appealing applications with location-based recommendations for users; governments can use this data to improve traffic mobility conditions and reduce air pollution; academia can utilise location data to gain deeper insights into fundamental societal issues, such as epidemiology [5].

Companies like Google, Apple, and Facebook, just to mention a few, collect, aggregate, share, and use personal information of their users. The accumulation of user data allows them to obtain almost unlimited knowledge about individuals, posing a risk to their privacy [6].

The mobile application development and data analysis industry have increasingly shaped the lives of users and their privacy. User profiling is often done, partially based on passive digital footprints created from data collected without users knowledge [7]. In addition to this aspect, applications rely on accessing sensitive user data (e.g., accounts, passwords, contacts, financial information, medical records, GPS, camera, and microphone) [8]. This data collection is different from active digital footprints created based on personal data provided by users consciously and voluntarily [9].

Article 4 of the General Data Protection Regulation (GDPR) defines personal data as information relating to an identified or identifiable natural person, and this information includes a wide range of data, such as full names, Internet Protocol (IP) addresses, addresses, device identifiers, and their location.

Regarding personal data, a problem arises in that many applications are not transparent about how personal information is used [10]. This problem is compounded by the fact that many applications request and share personal user information without providing any justification for collecting this data or how it is shared. An example of this type of event occurred in 2018 when the dating application *Grindr*, targeted at the gay community, shared Human Immunodeficiency Virus (HIV) status and user location with third parties [11].

The privacy policy establishes the legal framework for an application or organisation regarding the collection and sharing of data about a user. Despite its importance, the privacy policy is often perceived by users as being too lengthy to read [12] and too complex to understand [13].

Android is an Operating System (OS) designed for smartphones, tablets, and other types of personal devices. It was initially developed by the company Android Inc. in the earl's 2000s, and later acquired by Google in 2005 with the aim of developing a platform and OS for use in mobile devices [14]. The Android OS has dominated the smartphone market for over a decade, achieving a market share of 87.8% [15].

Given the high market share of the Android OS, there is a large number of applications available for users of devices that use this OS. The latest numbers indicate that there are 2.59 million applications available in the *Google Play Store* [16]. These numbers show that people recognise value in using the Android OS [17]. However, both the popular press and research have shown that the use of these applications poses serious security and privacy risks to their users [18] [19].

The privacy risks associated with location are of particular importance, as 74% of users use Location-based Service (LBS) [20]. According to a study conducted by Pew Research Center [1], approximately one-fifth of smartphone users (out of 2,254 participants) disabled location access features on their phones, expressing concerns about others or organisations accessing their location information [21].

In Android, this problem has been addressed by providing users with information about sensitive data access by applications. This information is provided when the application is installed, when the application first attempts to access sensitive data, and in some cases, whenever the application needs to access the user's sensitive data. Despite efforts by the owners of the Android platform, research in this domain has shown that this approach is not effective because the majority of users do not pay proper attention to the interfaces used to manage permissions [22] [23] [24].

Considering the aforementioned difficulties in controlling application access to location data, we propose a solution that allows the user to intuitively, dynamically, and interactively choose which installed applications can access their precise location.

[1]Pew Research Center is a non-partisan think tank based in the United States. Established in 2004, the Pew Research Center conducts surveys and studies on a wide range of social, political, and economic issues, providing valuable insights and data-driven analysis to inform public discourse and policy-making.

Our solution would entail implementing a mechanism to safeguard Android device users privacy by regulating installed applications access to their actual location. This feature would provide users with an improved granular control over location data access for each individual application, while simultaneously allowing them to provide fake location data to selected applications.

To offer the user the previously described features, our solution would encompass the following characteristics:

Enhanced control over location data access: The user would have a permission configuration interface for all installed applications on their device. This menu would allow users to view and manage location permissions for all installed applications in one central location.

Clear and user-friendly interface: The permission settings menu would have a clear and user-friendly interface, making it easy for users to understand and navigate.

Dynamic management of real location data access: Users would have the ability to dynamically manage true location data access for each application. They could grant or revoke access to their true location at any time, depending on their trust level or specific needs.

Generation of user-selectable simulated location data: The user can easily and intuitively generate fake location data at any time. This data can be *static* locations (choosing fixed points on the map) or *dynamic* (creating driving or walking simulated routes).

***Root* access to the device**: Acquiring administrative privileges on an Android device allows its users to have complete access to the system. This privileged access can be used for customization purposes, as well as to access features that are not available to regular users. Although this procedure is often discouraged by manufacturers, who do not assume warranty for *rooted* devices, on the other hand, there are applications designed to enhance Android security that only work with *root* access to the device (e.g., Titanium Backup, AdAway, AFWall, etc.).

1.1 Motivation

In the context of smartphones, location is considered to be the most sensitive data collected about the user [25]. Several studies conducted show that knowledge of the places visited (also known as Points of Interest (POI)) by a user allows for inferring various attributes of their profile [26] [27], as well as their social relationships [28] [29]. In addition to this aspect, if an attacker has the ability to infer or track the device's location, they can easily stalk its user [30].

In addition to the privacy risks associated with accessing location data related to the user's most visited places, the study referenced in [31] revealed that half of the Android applications studied (e.g. *Evernote* and *MySpace*) expose location data by sending it to advertising servers without users informed consent.

Privacy risks can also be heightened by the possibility of a malicious actor compromising a LBS application server [32]. An example of this threat was the discovery, in 2013, by a *black hat* hacker known as Peace, of information about over 167 million *LinkedIn* users [33], including location data. In 2016, Peace also accessed the data of over 360 million *MySpace* users [34].

The criticality of location data becomes even more evident as accessing it allows for sensitive information to be inferred about users, such as social relationships, health, religion, data related to nightlife and sexual life, among other aspects [35].

Another aspect that holds particular importance in this context is the fact that users demonstrate significant concerns regarding the privacy of their location data. This concern is clearly demonstrated in a study conducted by ISACA[1] [36] in which 90% of the study participants expressed concern about their location data being disclosed for advertising purposes, to strangers, friends, employers, and others. In the same study, only 6% of the participants showed no concerns about their location data, while the remaining 4% were uncertain about whether their location data is shared with third parties during the use of LBS.

[1]ISACA (Information Systems Audit and Control Association) is an international non-profit organisation that specialises in information security and governance.

1.2 Proposed Solution

The work carried out within the framework of this thesis aims to devise a *rooted* solution that provides Android mobile device users with precise control over how various applications access their true location data. This management should not compromise the normal functioning of the device or the installed applications.

It is essential that users can, at any time, grant or revoke access to their actual location for any application present on their device, while also having the option to choose fictitious location data to be provided to the applications instead of their real location. This process should be carried out without the need to revoke previously granted location access permissions to applications, allowing them to continue functioning normally, receiving fictitious location data chosen according to the user's preferences at any given time.

1.2.1 Objectives

The main objectives we aim to achieve with the proposed solution are as follows:

Literature review: The literature review aims to identify the current state of the art regarding research conducted in the context of the problem at hand. This aspect also provides valuable insights into the pros and cons of various approaches, helping to propose a more comprehensive solution considering the multiple aspects of the problem.

Solution design: The conceptualisation of the solution plays a crucial role as it provides a holistic view of the requirements necessary for implementation and how the various components will interact with each other.

Implementation: The practical translation of the solution design should allow for a *proof of concept* of the proposed architecture, providing a solution capable of addressing our problem.

Security and performance analysis: To assess the performance of our solution, it becomes necessary to test its ability to control application access to real location data by providing them with fictitious location data chosen by the user. These tests should be conducted with applications that utilise device location data and are widely popular among users. In addition to these aspects, it is also important to evaluate the impact of our solution on the functioning of the selected applications receiving fictitious location data, as well as its performance, particularly with regard to battery consumption.

1.2.2 Features

Our proposal presents the following functionalities:

Flexibility: The user should be able to choose, at any time, from the set of installed applications on their device, which ones will have access to their real location and which ones will be provided with a false location. In addition to being able to choose the applications to which fictitious location data will be provided, the user can also change the type of data at any time, enhancing the flexibility for customising fictitious location data.

Customizable: For the applications that will receive fake location data, users should be able to choose from the following options:

1. **Static location**: The user should be able to define *static* location points as their location. This can be done in two ways: by selecting a point on the map (manually placing a pin on the map) or by pressing a button to set their current location as a *static* point (even if the user moves to another location, their location will remain at the point where they pressed the button for this option).
2. **Simulated routes**: By opting for this mode of operation, the user should have the ability to create fictional routes that simulate car or pedestrian movements.

1.3 Contributions

With this work, we aim to improve the current state of the art by making the following contributions:

Literature review: The literature review allowed us to study various approaches to the problem identified in subsection 1.1 and understand the strengths and weaknesses of each solution.

Development of a functional prototype for Android devices: Based on the knowledge acquired through the literature review and related work, we have developed a functional prototype that enables the concealment of the user's true location and facilitates the capability to present a false location exclusively to selected applications, as determined by the user.

A *rooted* solution: Considering the advancements in *rooting* methods, obtaining *root* access on an Android device has become an easy and secure process [37]. This enables users to take full advantage of their devices, such as performing complete backups, removing *bloatware* [38], running paid apps for free [39], running apps on external memory, modifying user interfaces, running *background services*, *overclocking* hardware, and installing custom operating systems for receiving the latest updates [37].

Enhanced privacy and confidentiality of location data: Our implementation allows users to protect their location data from applications installed on their device by enabling them to define fictional location data of their choice at any time. Our solution offers privacy protection tailored to individual applications, with the degree of protection commensurate with the specific threat posed and the level of location detail required by each application.

Practical deployment and usage: Our solution offers a practical deployment and user experience, ensuring independent protection for each app without requiring any modifications to the existing applications and minimising user interaction. This innovative approach effectively safeguards the apps while maintaining a seamless user experience.

1.4 Outline

This thesis is structured into the following chapters: Chapter 2 presents the fundamental concepts necessary to address the issue under analysis. We begin by describing the essential components of the Android platform, followed by a brief explanation of the process of *rooting* Android devices and the frameworks, Application Programming Interface (API), and tools that underpin the implementation of our solution. In Chapter 3, we conduct a *state-of-the-art* review of proposed solutions to tackle the issue of user privacy, specifically concerning the control of access to real location data by applications installed on Android devices. Chapter 4 elaborates on the architecture of our solution, and Chapter 5 provides a detailed account of the implementation process of our proposal. In Chapter 6, we present the results of the tests conducted on our solution. Chapter 7 presents suggestions for future work and concludes with some final considerations on the completion of this dissertation.

Chapter 2

Background

In this chapter, we present some fundamental concepts for approaching the problem at hand. We begin by describing the core components of the Android platform, with a particular focus on the various elements that constitute the Android OS. This is followed by a brief overview of the process of *rooting* an Android smartphone, the current permission model in use, and the compilation process of an Android application. Subsequently, we provide a comprehensive description of the concept of *code reflection* and the *Java Reflection* API.

The *Java Reflection* API serves as a vital tool in the operation of the *EdXposed* framework, which is employed in the development of the proposed solution.

To shed light, albeit in a generic manner, on the main components shaping our solution, we provide a description of the *Xposed* framework and one of its latest versions, the *EdXposed* framework, along with the *Magisk* tool and its *Riru* module, which are employed for device *rooting* and supporting the operation of the *EdXposed* framework.

This chapter concludes with a brief account of the process used for executing *Lua scripts* in Android applications and the *Lua-to-Java* interpreter known as *LuaJ*.

2.1 Android Platform

The Android OS was designed for smartphones, tablets, and other types of personal devices. It was initially developed by Android Inc. in the early 2000 and later on it was bought by Google in 2005, which furthered its development as an operating system and as a platform for mobile devices [14]. This mobile OS, based on Linux, consists of several layers, as shown in Figure 2.1.

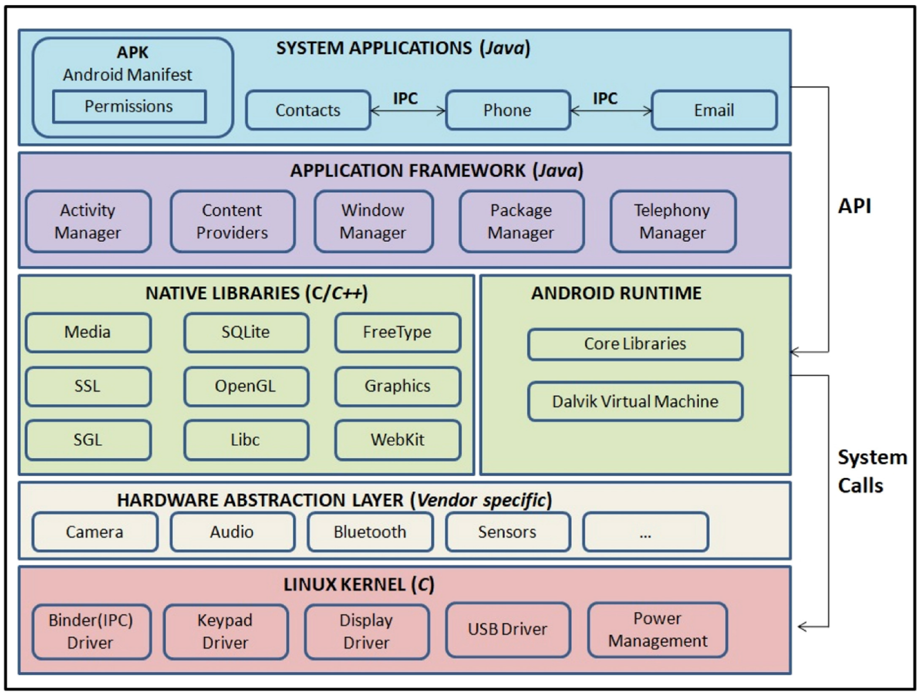


FIGURE 2.1: Architecture of the Android platform
Source: Adapted from [40]

The architecture of the Android platform was conceived with a logic of simplifying component reuse. Services in modular components allow applications to share their resources among themselves. Android also allows components to be replaced by the user [40].

The architecture of the Android framework can be seen as a layered system in which each layer has the following functionalities:

Kernel - It is responsible for providing support and management for essential system services such as memory, security, networking, etc.

Hardware Abstraction Layer (HAL) - This layer serves as an interface for communication between the Android application/framework and the specific hardware drivers of the device, such as the camera, Bluetooth, etc. The HAL is specific to each hardware and its implementation varies depending on the vendor.

Android Runtime (ART) - ART was introduced as a new runtime environment in Android versions later than 5.0. During the application installation process, it uses a combination of Ahead-of-time (AOT) and Just-In-Time (JIT) compilation techniques. AOT compilation converts *Dalvik bytecode* into native binaries Executable and Linkable Format (ELF), while JIT compilation optimises garbage collection and power consumption, resulting in improved execution performance.

Native libraries - The core system services and various components of Android, such as ART and HAL, are built using native libraries written in *C/C++*. There are different libraries, which provide support in building user interface application framework, drawing graphics and accessing database.

Application framework - The Android Software Development Kit (SDK) provides a comprehensive set of tools and API libraries to support the development of Android applications. This framework, known as *Android Application Framework*, offers several essential features, including a database for storing data, support for various audio, video, and image formats, debugging tools, and more.

System applications - Applications are located at the topmost layer of the Android stack. These can include both native and third-party applications such as web browsers, email clients, Short Message Service (SMS) messengers, etc., which are installed by the user.

2.1.1 Rooting

Rooting is a process that grants device users persistent privileged control, commonly known as *root* access, over their devices. The most widely accepted method to achieve this persistent *root* access involves installing a custom *su* binary, also referred to as *switch user*, *super user*, or *substitute user*. This custom binary enables any application on the device to execute privileged operations as *root* (as discussed in Section 1.1).

Once the device has been successfully *rooted* (i.e., with the custom *su* binary installed), users gain the ability to bypass restrictions imposed by carriers and hardware manufacturers. They can modify or uninstall system applications, and leverage various *root* applications that necessitate elevated privileges.

A good example of the use of *root* solutions is the popular community-built system images (e.g.,*LinegaOS* [41]) and *root* apps (i.e., apps that require *root* access such as *Titanium Backup* [42], *Root Explorer* [43]) each has over 10 million downloads.

Although useful to device owners, *rooting* may weaken the security of Android devices. The security of a *rooted* device relies solely on the device user regulating *root* access properly. Yet, the research shows that many users ignore security warnings due to habituation or lack of contextual information [44] [45].

Despite the security issues posed by the *rooting* process, it is observed that in certain contexts, the number of *rooted* devices is significantly high. According to a *Google security report* on Android, conducted in 2014, [46], *Google Verify Apps* detected *rooting* apps (i.e., apps that *root* the devices via privilege-escalation vulnerabilities) installed on approximately 2.5 M devices, and particularly about 34% of Chinese devices have a *rooting* application installed. Note that the number does not include other *rooting* methods (e.g., unlockable bootloader, bootable SDCard, OEM flash utilities). Moreover, *Verify Apps* found that there are numerous applications from major Chinese corporations that include *rooting* exploits to provide functionality that is unavailable through the official Android API [47].

In addition to the aspects mentioned earlier, it is important to note that in a survey conducted by Tencent [48] in 2014, 80% China smartphone user respondents *rooted* their smartphones (see Figure 2.2).

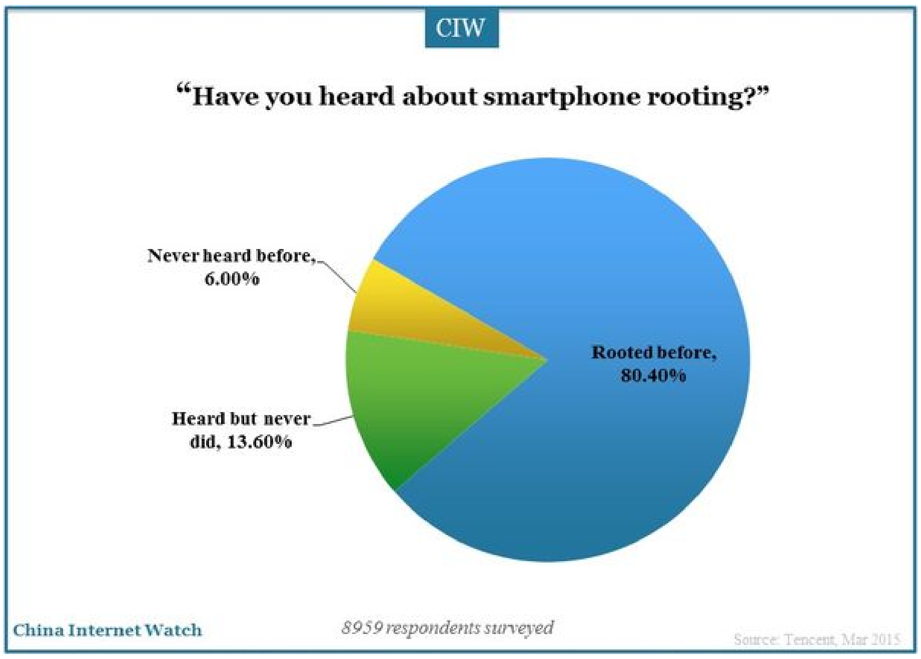


FIGURE 2.2: Percentage of users who rooted their Android devices in China in 2014
Source: Adapted from [48]

2.1.2 The Permission Model

Individuals and organisations frequently depend on requested permissions linked to mobile apps, along with official privacy policies, to assess the safety of an app and ascertain the type of information being collected [17].

At the API level, every app needs to request permissions to access various resources on the user's device [49]. App developers are required to declare this set of permissions in their `AndroidManifest.xml` file in order to access the corresponding API [50]. In this manner, every application must specify the functionality and sensitive data it requires when users attempt to install them. However, it is not evident to the users how the granted permissions for accessing sensitive data on their devices will be utilised once these applications are installed [51].

Aware of the limitations of the initially implemented permissions model, Google starting from Android 6.0 Marshmallow, also enabled applications to request permissions to access sensitive user data at runtime, rather than solely during the installation process [52].

Accessing application permissions statically does not enable the user to have a realistic perception of how these permissions impact their privacy. An application can access sensitive user data that the user would not permit if they were aware of the risks that specific application poses to their privacy [14]. For instance, a map application may require access to the user's location in order to provide real-time directions. However, once granted access to the location, the application may transmit the user's location to a remote server, potentially impacting their privacy in various ways [53]. The user remains unaware of how the map application handles their location information.

Merely presenting the user with a list of required permissions does not adequately inform them about how granting those privileges can impact their privacy. As a result, their privacy can be compromised through various means [54].

2.1.3 Project Building

The process of building an Android application is a highly intricate procedure that encompasses various tools. Initially, the resource files undergo compilation and are referenced within an `R.java` file. Subsequently, the Java code is compiled and transformed into Dalvik bytecode using the *dex* tool. These files are then assembled into an Android Package (APK) file (see Figure 2.3), which is subsequently signed with either a debug or release key. Finally, the application can be installed on a device [55]. Engaging in each of these steps manually would entail a laborious and time-intensive endeavour.

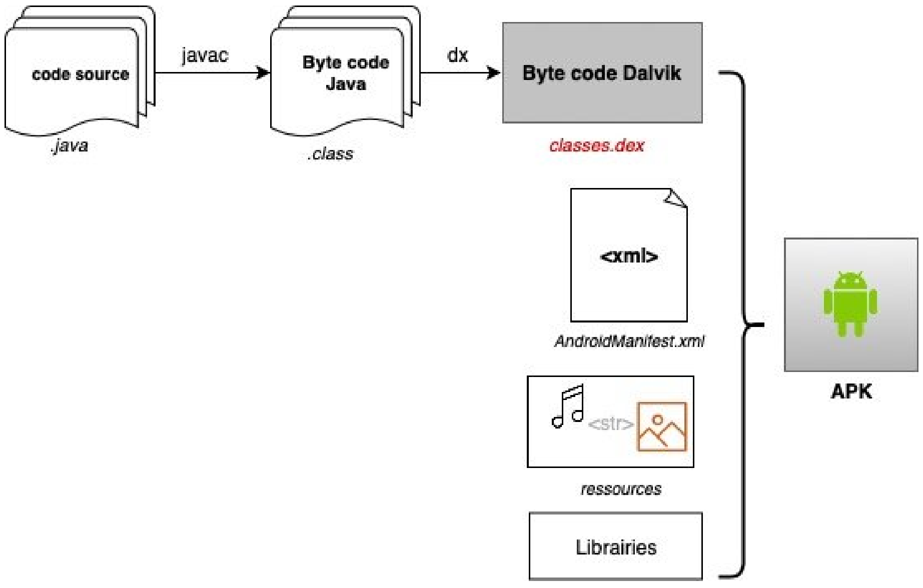


FIGURE 2.3: Compilation process of an Android application
Source: Adapted from [43]

Taking into consideration the aforementioned, Google has made an effort to simplify the Android application development process as much as possible. This has been achieved through the creation of tools, such as *Android Studio* and *Gradle plugin*, that facilitate and streamline developers work.

2.2 The Java Reflection API

Reflection is a prominent aspect of the Java programming language, providing the capability for an executing Java program to scrutinise or introspect its own structure and modify internal program properties. This powerful feature enables a *Java class*, for instance, to retrieve and exhibit the names of all its constituent elements [56].

The use of code *reflection* provides us valuable insights into the class to which an object pertains and further provides us with information about the methods within that class that can be invoked using said object. By leveraging *reflection*, we gain the ability to dynamically invoke methods during runtime, regardless of the access specifier associated with them [57].

2.3 Xposed Framework

The *Xposed* framework adheres precisely to its name, serving as a software component that facilitates the override of generic code with user code to extend or modify functionality. In essence, it *exposes* the system's code to the framework, necessitating *root* privileges, and grants users the capability to integrate diverse modules that introduce varied or supplementary modifications without modifying any APK [58] [59].

2.4 EdXposed Framework

With the advent of the Android Pie (version 9, SDK 28), Google introduced a series of modifications aimed at bolstering the security of the OS, including changes to *SELinux*[1] policies. These alterations resulted in the *Xposed* framework ceasing to function on devices running this and subsequent versions of Android. To overcome this difficulty, the *EdXposed* framework emerged as an extended version of the *Xposed* framework, which is compatible with all Android versions from 8 to 10.

EdXposed is a *Riru* [2] module designed to offer an ART *hooking* framework that maintains consistent API with the original *Xposed*. It boasts key attributes such as being open-source, stable, minimally invasive, and remarkably fast [60].

Under the hood, *EdXposed* utilises the *SandHook* [3] and Yet Another Hooking Framework for Android (YAHFA) [4] [61] projects.

EdXposed framework includes its own companion app called *EdXposed Manager*, which allows users to monitor the core's status and download various *Xposed* modules [62].

2.5 Magisk

In essence, *Magisk* is a versatile tool designed to grant *root* access to Android devices, resembling conventional utilities like *SuperSU*, yet its capabilities extend far beyond mere

[1]*SELinux*, which stands for *Security-Enhanced Linux*, is a security mechanism implemented in the *Linux kernel* to enforce Mandatory Access Controls (MAC). It provides an additional layer of security beyond the traditional Discretionary Access Controls (DAC) commonly used in operating systems.

[2]We will address the operation of the *Riru* framework in the Section 2.5.1

[3]*SandHook* is a *hooking* framework used in the context of the *EdXposed* framework, which allows for the interception and modification of Android application behaviour. *SandHook* works by intercepting and modifying the behaviour of Android applications at specific *hook* points, enabling users to customise and enhance their Android experience through the use of *Xposed* modules.

[4] YAHFA is a *hooking* framework used in Android app modification. It facilitates the interception and customization of Android application behaviour, often in conjunction with the *EdXposed* framework and *Xposed* modules.

root access. It is the brainchild of XDA *Senior Recognised Developer*, *topjohnwu*, and serves as a gateway to a wide array of modifications for Android phones.

Alongside providing *root* access, *Magisk* offers an abundance of derivative components known as *Magisk modules*, which can be installed for various purposes [63].

Magisk offers an innovative *systemless* approach, distinguishing it from the *Xposed* framework, which involves direct modifications to the Android system, thereby triggering *Google SafetyNet* to disable services like *Google Play*, *Netflix*, and *Pokemon GO*. In contrast, *Magisk* adopts a non-intrusive approach by refraining from altering the system directly.

Magisk has garnered widespread popularity due to its unique *systemless* approach to Android modification. Unlike conventional methods that involve direct alterations to system files, *Magisk* operates by utilising the boot partition, ensuring that the core system remains untouched.

When the operating system requests a system file, *Magisk* seamlessly overlays a virtual file in its place, preserving the original system file intact and unaltered. This is crucial for users who wish to receive Over-The-Air (OTA) updates and utilise apps that are safeguarded by *Google SafetyNet*, as any tampering with the core system could result in the loss of these privileges [64].

When installed through *Magisk*, the *EdXposed* framework provides the ability to bypass *SafetyNet*, particularly the Google Mobile Services (GMS) and Google SafetyNet Attestation API (GSG) services.

2.5.1 Riru Module

Developed by *Rikka* and *yujincheng08*, *Riru* is a meticulously designed *Magisk* module that offers functionalities similar to *Xposed*, thereby eliminating the need to install the conventional *Xposed Framework*. Its implementation involves injection into *Zygote*, enabling the execution of codes from various modules within apps or the system server [65].

In its initial implementation, *Riru* relied on the substitution of a specific system library known as *libmemtrack*. This approach was abandoned in favour of utilising a system property referred to as the *native bridge* (`ro.dalvik.vm.native.bridge`). By leveraging this property, developers can dynamically load and unload shared libraries according to their preference, thereby facilitating the injection procedure into the *Zygote* process.

Riru functions as a gateway, facilitating the integration of other modules with the *Zygote* process. Consequently, for the incorporation of Riru-compatible modules, they must be installed through the *Magisk* app, following the same procedure as any other *Magisk* modules. Once successfully installed, *Riru* modules will be displayed alongside *Magisk* modules within the *Magisk* app.

2.6 Programming Language Lua

The Lua language was created by Roberto Ierusalimschy, Waldemar, and Luiz in 1993. Originally developed as an *in-house* language for two specific projects, Lua has now become extensively utilised across various domains that can derive advantages from a adaptable, portable, and efficient scripting language. These domains encompass embedded systems, mobile devices, the Internet of Things, and, naturally, the realm of gaming [66].

Lua combines a straightforward procedural syntax with robust data description constructs that rely on associative arrays and extensible semantics. It is a dynamically typed language, functioning through bytecode interpretation for a register-based virtual machine, and incorporates automatic memory management featuring incremental garbage collection.

Taking into account the listed features, the Lua programming language proves exceptionally well-suited for configuration, scripting, and rapid prototyping purposes. Furthermore, Lua is freely available as open-source software, distributed under the widely recognised and permissive MIT license. Users are granted complete freedom to utilise Lua for any purpose, including commercial endeavours, without incurring any cost [67].

2.6.1 Embedding Lua in Android Applications

Considering the aspects of the Lua language previously mentioned in section 2.6, such as its speed and runtime performance, it emerges as a compelling solution for executing scripts in *Java-based* Android applications.

The extensibility of Lua provides developers with the opportunity to incorporate custom extensions and native libraries, thus enhancing application functionalities. Additionally, its portability enables development for multiple platforms and code reuse, making it an ideal scripting language for optimizing performance while consuming fewer resources, ultimately enhancing the user experience of Android applications.

2.6.2 Luaj - A Compact Lua VM Written in Java

LuaJ is a Lua interpreter written in Java, boasting an incredibly small size – when obfuscated, it amounts to less than 25 KB. This reduction in size stems from its development's objective to overcome certain limitations imposed by Java 2 Platform, Micro Edition (J2ME)[1], particularly the absence of a *ClassLoader*.

LuaJ stands as a robust virtual machine, augmented by the inclusion of standard libraries, totalling 50 KB. Subsequently, a compiler was incorporated, adding an additional 25 KB, thereby enabling the execution of any valid Lua script. To complement these components, a debugging library of approximately 25 KB was added, completing the scope of this project [68].

The latest version of *LuaJ*, version 2.0, represents a significant rewrite, strategically leveraging the *Java stack* instead of a separate *Lua stack*. Consequently, *LuaJ* can be utilized both with and without an actual Lua bytecode interpreter. This approach facilitates the direct compilation of Lua bytecode into Java bytecode, the conversion of Lua source code into Java source code, or the interpretation of Lua bytecode.

Flexibility is paramount, allowing seamless adaptation to the needs of the host runtime environment [68]. Given its characteristics, this interpreter proves to be a viable solution for executing Lua scripts within Android applications written in Java.

[1]J2ME is a platform developed by Sun Microsystems (now Oracle Corporation) for developing mobile and embedded applications using the *Java* programming language. J2ME was particularly popular for developing applications for feature phones and other resource-constrained devices.

Chapter 3

Related Work

Among the sensitive data collected about the user by various applications installed on the device, as well as *location-based services*, location stands out as the one perceived to have the greatest impact on users privacy [69].

In this Chapter, we present several solutions that address this issue. We begin by examining various implementations, aiming to identify the strengths and weaknesses of each, in order to pinpoint the most relevant aspects that will enable us to approach the problem more efficiently.

3.1 Research Methodology

In order to analyse similar solutions, we have established a set of keywords based on the functionalities defined in Section 1.2.2. The set of keywords used in the research was as follows:

Android	Privacy Enhancement
Instrumentation	Permission Manager
Sensitive Information	Monitoring
Security	Location privacy preservation
Personal Information	Location-based service

The search engine used in the research was *Google Scholar*, and searches were also conducted using the *Google search engine* for complementary information.

Taking into account the evolution of the Android platform, only those solutions that provide added value to our research from a conceptual and practical perspective were considered. This is the case, even though some of them are no longer feasible for practical use.

3.2 Similar Solutions

The protection of user location data on Android devices has been a subject extensively studied by the academic community, particularly in the last decade. This prevailing trend is evident in the vast literature dedicated to this theme. It is crucial to emphasise that the solutions presented herein pertain solely to studies with a specific focus on safeguarding user privacy concerning location data.

As previously stated, users perceive this data as the most compromising to their privacy when accessed in an abusive or illegitimate manner. With this aspect in mind, this section presents the solutions deemed most relevant to our research, accompanied by a concise description of their functioning.

Jeon et al. [70] propose an *unrooted* solution based on a set of tools referred to as *RefineDroid*, *Mr. Hide*, and *Dr. Android*, whose purpose is to enable application developers and their users to comprehend, assess, and apply detailed permissions on standard Android applications and devices. The Figure 3.1 provides an overview of this toolset and how its components interrelate.

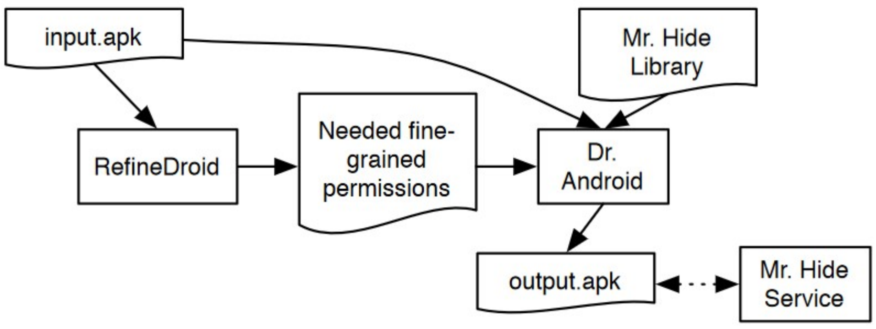


FIGURE 3.1: Overview of fine-grained permissions framework
Source: Adapted from [70]

This solution requires each application to undergo to an adaptation process, during which the comprising tools offer the following functionalities:

RefineDroid - At the time of its development, this tool was capable of inferring five types of fine-grained permissions utilized by Android devices, including access to the Internet, user contacts, and system settings. For instance, this tool can infer permissions in the form of `InternetURL(d)`, which grants access solely to the domain `d`.

Mr. Hide - This tool consists of a set of Android services that encapsulate various privileged Android APIs and dynamically apply a set of fine-grained permissions created by the authors of this solution. Furthermore, this tool provides a `hidelib` library responsible for all communication with *Mr. Hide*, serving as a direct substitute for the most sensitive Android APIs.

Dr. Android - *Dr. Android* (*Dalvik Rewriter* for Android) is a tool designed to remove Android permissions from existing application packages, replacing them with specific versions of fine-grained permissions that are validated by the *Mr. Hide* tool.

This solution, although conceptually valid, has the disadvantage of relying on developers to adapt their applications to the finer permissions proposed by the authors. Although *Dr. Android* and *Mr. Hide* can run on unmodified Android devices, they rely on the developers willingness to use this framework API. Furthermore, to function properly, application modification is necessary, which significantly complicates the process of installing a new application [70].

Fawaz et al. [69] propose a framework, named *LP-Guardian*, for safeguarding the privacy of Android device users. Despite the fact that this solution was developed for the Android platform, it is also applicable to other platforms where access to location data is managed through a permissions system (e.g., Windows Phone, BlackBerry OS), as well as those that manage access to location data by obtaining explicit user authorisation for each access (e.g., iOS). The operation of *LP - Guardian* is only possible on *rooted* devices or through the use of a custom Read-Only Memory (ROM).

In the block diagram shown in Figure 3.2, we can observe the components of *LP-Guardian* and their interactions. According to the authors, *LP-Guardian* protects the user's privacy at three distinct levels. It tackles tracking threats by reducing the amount of time tracked per day. It addresses the profiling threat by allowing users to conceal sensitive locations. Finally, it mitigates identification threats through an innovative mechanism that renders the user's mobility pattern indistinguishable.

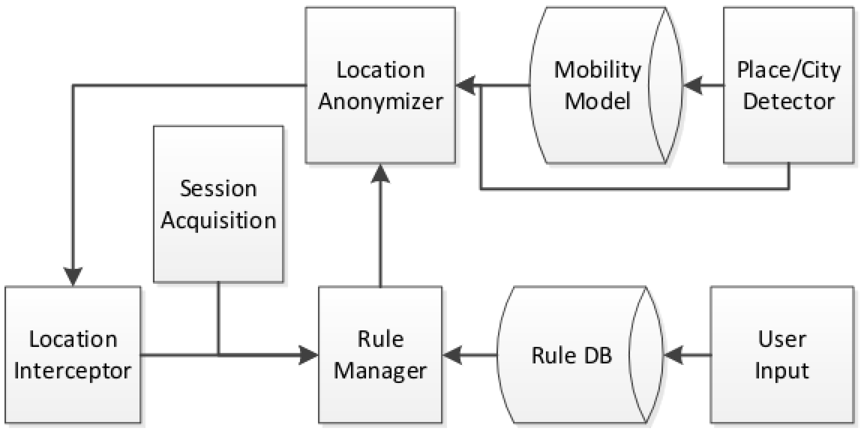


FIGURE 3.2: LP-Guardian's architecture and interactions of its components
Source: Adapted from [69]

From a conceptual standpoint, this approach is very similar to the one implemented as a *proof of concept* in this dissertation. However, it falls short in providing greater flexibility to the user regarding the creation of *synthetic* routes, following a case-by-case logic based on their privacy perceptions during the use of their Android device.

Beresford et al. [71] propose a solution, which they named *MockDroid*, consisting of a modified version of the Android operating system. This modified version enables a user to *mock* an application's access to a resource, subsequently resulting in the resource being reported as empty or unavailable when the application requests access. With this implementation, an application never receives location updates, leading to a solution that allows for complete privacy but, from a practical standpoint, offers no usefulness.

Enck et al. [31] propose a solution known as *TaintDroid*, which serves as an extension to the Android mobile-phone platform with the primary objective of tracking the flow of privacy-sensitive data through third-party applications. This system, based on a custom ROM, operates on the assumption that downloaded third-party applications cannot be inherently trusted. Consequently, *TaintDroid* undertakes real-time monitoring of these applications, focusing on how they access and manipulate users personal data.

The core objectives of *TaintDroid's* implementation are twofold: firstly, to identify instances where sensitive data exits the system through untrusted applications, and secondly, to facilitate comprehensive application analysis for users and external security services.

TaintDroid employs a labelling mechanism that designates privacy-sensitive data, also referred to as *taint source*, at various levels, including variable, method, message, and file. Subsequently, the system tracks the flow of this *tainted* data throughout the entire system. When this marked data attempts to exit the system through a network interface, recognised as a *taint sink*, *TaintDroid* promptly provides real-time feedback to users. This feedback encompasses information concerning the data being transmitted, the application involved, and the destination to which the data is intended.

The usage of these concepts in the *TaintDroid* architecture is graphically depicted in Figure 3.3, which illustrates the data flow from the *source* to the *sink*. The visual representation showcases how the system tracks and discerns the movement of privacy-sensitive information.

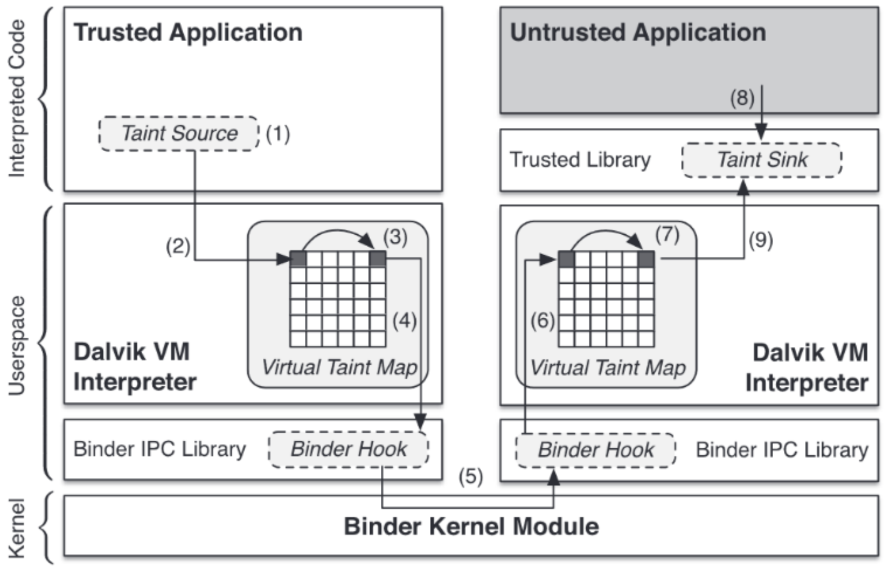


FIGURE 3.3: TaintDroid architecture within Android
Source: Adapted from [31]

This solution embodies a valid conceptual perspective for addressing our issue, albeit lacking any functionality aimed at preventing unauthorised access to sensitive user data. Moreover, even if employed solely for monitoring purposes, *TaintDroid* incurs an excessively high cost in performance, potentially reaching up to 30%, thereby significantly and adversely impacting the user experience.

Fawaz et al. [72] present *LP-Doctor*, an application designed at the user level with the purpose of safeguarding the privacy of Android smartphone users location data.

LP-Doctor proactively assesses the potential privacy risks posed by an application before launching it. If launching the app from the current location presents any privacy risks, *LP-Doctor* intervenes to protect the user's privacy. Additionally, it discreetly notifies the user of such potential risks.

This solution effectively serves as a control mechanism for the underlying tools of the operating system. Furthermore, *LP-Doctor* empowers users to finely tune the privacy-utility trade-off for each application, enabling them to adjust the protection level while the app is in use.

The authors integrated *LP-Doctor* with *CyanogenMod's app launcher* [1]. It operates as a *background service*, intercepting app-launch events, making informed decisions regarding suitable actions, executing these actions, and subsequently instructing the app to launch. The high-level execution flow of *LP-Doctor* is illustrated in Figure 3.4.

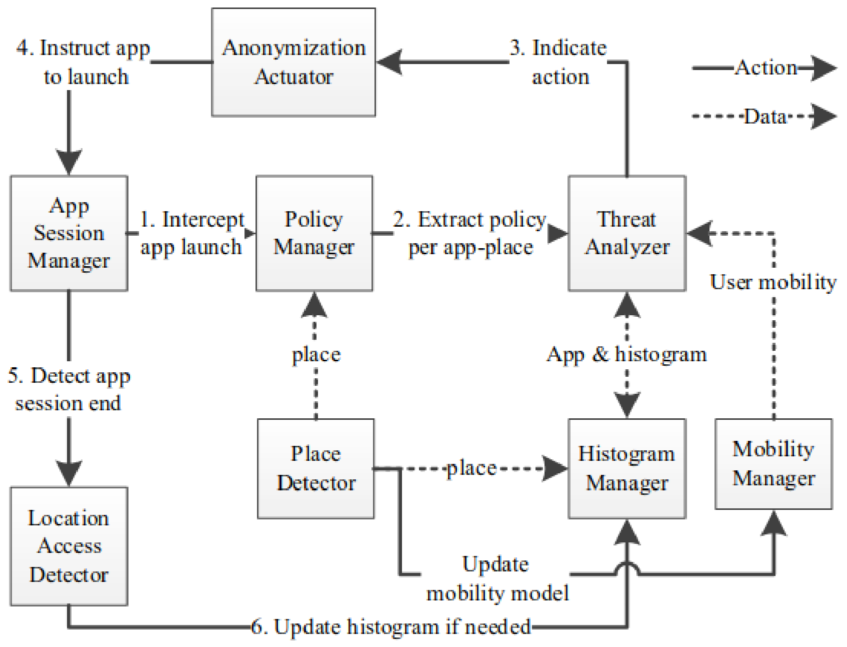


FIGURE 3.4: Execution flow of LP-Doctor when a location-aware app launches
Source: Adapted from [72]

LP-Doctor engages in user interaction to relay privacy-protection status. Additionally, it empowers the user to configure privacy profiles for various applications and locations.

This solution represents an extremely effective way to raise awareness among users about the efficient use of the controls already available in the operating system. However, it proves inefficient in the objective of providing false location information to a specific application without the application being able to verify whether the location data it is receiving is real or fake. This is due to the fact that this solution utilises the mock location provider, which is a developer option provided by the operating system.

[1] *CyanogenMod's app launcher* refers to a software component, developed by the *CyanogenMod* community, that provides a user interface for accessing and organising applications on Android-based devices.

This solution is also ineffective for applications that require precise location access, such as navigation apps. Additionally, it is entirely inefficient in controlling location data access for *background-running* applications.

Hornyack et al. [73] have devised a sophisticated system known as *AppFence*, which retrofits the Android operating system to enforce privacy controls on pre-existing (unaltered) Android applications.

Through *AppFence*, users are granted the capability to withhold sensitive data from imperious applications that inappropriately seek unnecessary information beyond the scope of their advertised functionalities. Additionally, for data that are legitimately indispensable for user-desired functions, *AppFence* strategically prevents any communications initiated by the application that might attempt to illicitly exfiltrate these valuable data from the device.

When an application requests access to sensitive data that a user prefers not to grant, *AppFence* adeptly replaces such information with innocuous shadow data, ensuring that the application remains oblivious to the actual sensitive content while still functioning appropriately.

AppFence incorporates a supplementary data-egress control mechanism to effectively safeguard authorised data from improper usage and unauthorised transfer outside the device, a process known as *exfiltration blocking*.

The authors have further expanded upon the *TaintDroid* information flow tracking system to encompass data derived from information that the user deems private. Consequently, *AppFence* can intercept and prevent undesirable transmissions of these classified data. Notably, for each specific sensitive data type within the system, *AppFence* can be conveniently customised to intercept and block messages containing such data. For a visual depiction of the *AppFence* system architecture, refer to Figure 3.5.

This solution presents a compelling approach, as it safeguards against unauthorised access to location data by certain more intrusive applications while also limiting the exfiltration of user location data from the device when such action is unnecessary for the normal functioning of the application.

The primary drawback of implementing this solution lies in the need to modify the OS, as current security measures implemented by Google (e.g., *SafetyNet*) could potentially compromise the device's functionality. Moreover, this solution lacks any customization options and does not address the issue of providing fictitious data to applications,

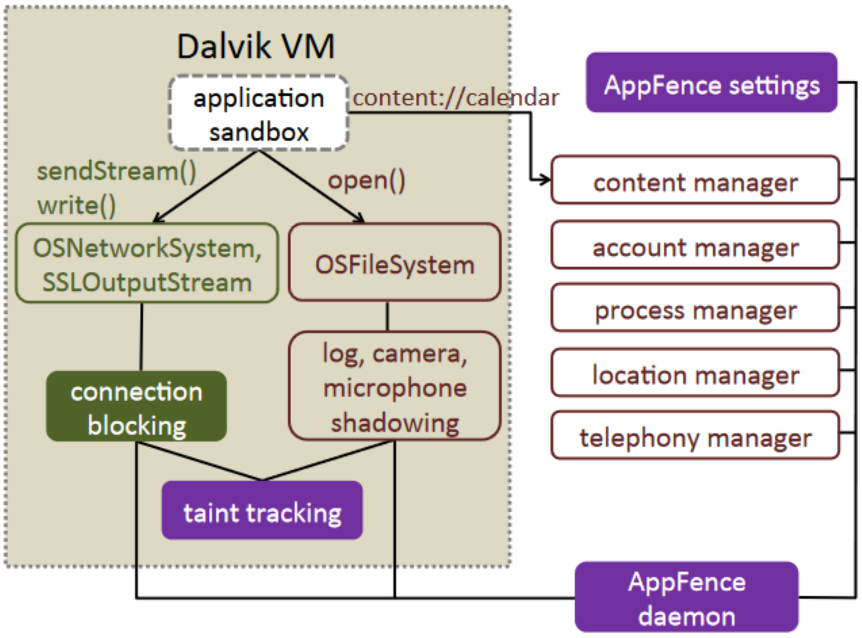


FIGURE 3.5: AppFence system architecture
Source: Adapted from [73]

which may lead to abnormal functioning in the majority of applications reliant on user device location.

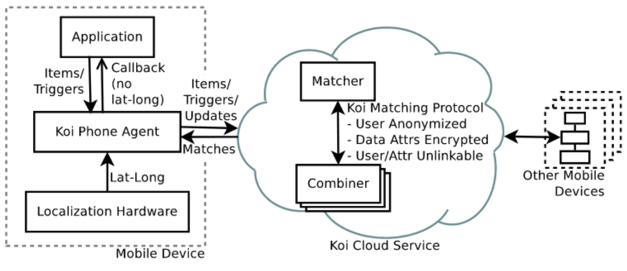
Guha et al. [74] introduce the concept of *privacy-preserving location-based* matching as a foundational platform primitive and offer it as an alternative to the direct exposure of low-level *latitude-longitude* coordinates to applications.

This platform, named *Koi*, consists of two fundamental components: one that runs on the user's mobile device, and the other in the *cloud* (as shown on the left-hand side of Figure 3.6).

The mobile component of *Koi* serves as an interface between applications and the cloud component. For applications, it offers a straightforward API (shown on the right-hand side of Figure 3.6), enabling registration and updates of *items* and *triggers*, and providing notification through callbacks.

An *item* represents a factual statement providing information about a specific entity, such as a user, a business, or a vehicle (e.g., bus). These statements consist of one or more *attributes* pertaining to the entity, including its geographical location.

The *location attribute* holds particular significance, as it can be automatically updated by *Koi*, as the user moves, for example, providing real-time location information. *Triggers*,



(a) Architecture

Registering Items and Triggers
ITEM = CREATEITEM(CONTENT, TTL)
TRIG = CREATETRIGGER(CALLBACK, TTL)
Create a new item or trigger.
Adding Attributes
ADDLOCATTR(ITEMORTRIG, LOC, AUTOUPDATE)
ADDAPPATTR(ITEMORTRIG, ATTR)
Associate location or other attributes to items and triggers.
Notifications
CALLBACK.NOTIFY(CONTENT)
Called by platform when a match is found.

(b) API

FIGURE 3.6: Koi architecture
Source: Adapted from [74]

on the other hand, share similarities with *items* but serve as queries, specifically requesting a callback when a matching condition is met.

The mobile component of the *Koi* system interacts with the cloud component by engaging in communication to register *items* and *triggers*, as well as to set and update their respective *attributes*.

The *Koi* cloud service comprises two *non-colluding* sub-components: the *matcher* and the *combiner*. Broadly speaking, the *matcher* possesses knowledge of user identities and their *attributes*, including location. However, it remains unaware of the association between users and their locations or other *attributes*.

To grant privacy preservation, the authors implemented a protocol to allow the *matcher* and the *combiner* to conduct matching operations without acquiring information about the association between users and their locations or any other *item* identities and *attribute* relationships. Nevertheless, the *matcher* can utilise the knowledge of plaintext location and other attributes to perform comprehensive and semantically-meaningful matching, encompassing geocoding, location proximity, or spelling correction, among other aspects.

On the contrary, the *combiner* possesses knowledge about the association between anonymized users and encrypted locations, as well as other *attributes*. However, it remains unaware of the actual identities or attribute values.

The *Koi* service can be employed by multiple applications. To avoid naming conflicts, attribute names are appropriately name-spaced according to the application responsible for registering the item.

While this solution is innovative and ensures that only the application and the user can access sensitive data, preventing LBS from establishing a link between the user and location data, it lacks mechanisms to enable specific applications to function without requiring access to the device's actual location.

Depending on a *cloud service*, the use of this solution will always be dependent on an Internet connection, which may not be feasible for some users with limited data subscription plans.

In addition to the previously mentioned aspects, the interaction among the various components that constitute this solution may cause delays in the operation of certain applications, leading to a negative impact on the user experience. This solution also requires that applications start using the *Koi* framework's API to access location data, rather than using Google's *lat-long* location API.

Kang et al. [75] proposed a solution for location-privacy preservation called *MoveWithMe*, which requires *root* access to the device to function properly. The authors named it *MoveWithMe* because it automatically generates a number of *decoys* to move with the user like real human beings and serve as distractions to the service providers.

In the *MoveWithMe* system, each *decoy* has its own moving patterns, favourite places, daily schedules, social behaviours, etc. Based on the user's privacy needs, the initial number of *decoys*, the *decoys* social and travel patterns, and their personalised profiles can be varied.

Figure 3.7 provides an overview of how the components in the *MoveWithMe* system collaborate with each other and interact with *location-based services*. In particular, to obtain the protection from *MoveWithMe*, the smartphone user simply needs to open the *MoveWithMe* app before visiting any *location-based* service websites.

If the service monitor detects that a *location-based service* requires the user's phone to upload the user's location information, the *MoveWithMe* app will automatically send a mixed group of the real user request and fake requests based on the *decoys* locations to confuse the service provider.

This solution appears to be effective in obfuscating the true device location concerning *location-based service* providers. However, it does not offer any user customization options for the *decoys*. Additionally, if the number of *decoys* is limited, the *location-based service* may still infer the actual user location, as this information is also transmitted. Moreover, the *MoveWithMe* application relies on an Internet connection to access *location-based services* and coordinate the *decoys*, which may not always be feasible or desirable for certain users. Furthermore, this solution does not enable concealing the true device location from other installed applications on the user device.

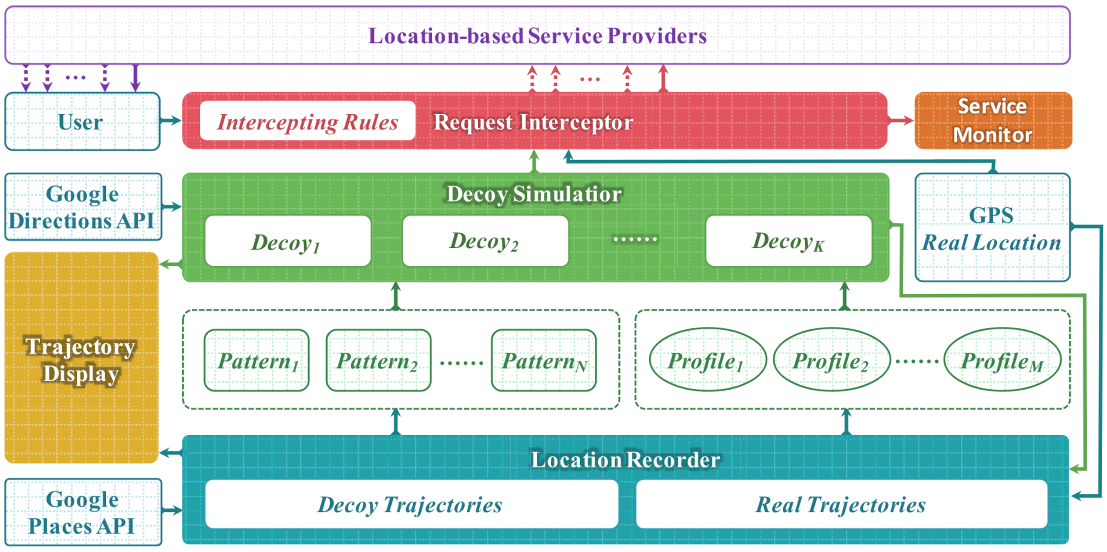


FIGURE 3.7: The Framework of the MoveWithMe System
Source: Adapted from [75]

Chitkara et al [76] present the design and implementation of Protect my Privacy (PmP) app for Android. The PmP which requires a *rooted* device, supports a feature of both App and library-based privacy control for accesses to sensitive data.

The figure Figure 3.8 illustrates the comprehensive architecture of the PmP app, offering insights into its various components and control flow. To comprehend why a specific app accesses sensitive data, the authors capture the stack trace whenever a user makes a decision (allow, deny, or fake) pertaining to privacy-sensitive data elements (e.g., location, contacts, identifiers, etc.).

The PmP application for Android empowers users to exercise control over their private data by granting, denying, or simulating access to the App or third-party libraries. The authors have devised a scalable backend system, facilitating the collection of user decisions and runtime stack traces for the detection and categorisation of libraries.

The fundamental goal of PmP revolves around deducing contextual information related to data accesses, specifically identifying the entity within the app that accesses sensitive data and the purpose behind it. This contextual understanding aids in the design of more effective privacy protection mechanisms.

The PmP Android app is not solely focused on identifying third-party libraries. It also empowers users with the ability to dictate which data is shared with these libraries. The solution further incorporates a scalable backend, tasked with aggregating, storing,

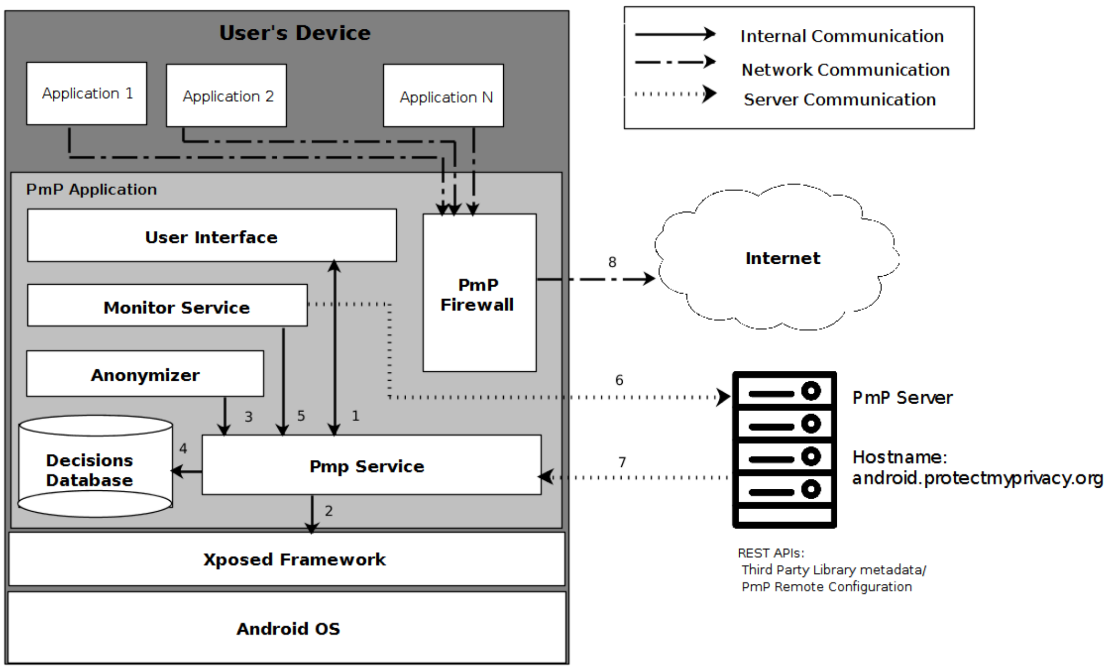


FIGURE 3.8: The architecture of PmP
Source: Adapted from [76]

and analysing diverse data points sent by PmP concerning user decisions, stack traces, and results from *in-app* surveys.

For anonymizing location data, the authors adopt a technique that transforms the input location into a coarser representation, essentially providing the location at the city-level granularity rather than a specific location. This approach strikes a balance between privacy preservation and enabling location-based functionalities to operate effectively. Additionally, the user has the flexibility to select any location as the fake location, utilising a user-friendly Maps interface.

PmP incorporates a *firewall* to impede application access to the Internet. This *firewall* service operates in the background, receiving callbacks from the *Xposed* framework whenever any app attempts to access private data. Additionally, a server is employed to store user decisions in an anonymous manner.

The fundamental basis of our proposal, which was limited in its implementation to the development of a functional prototype as a *proof of concept*, relies on the same criteria as this solution that addresses the primary concerns regarding user location data access by both installed applications and the libraries they utilise.

As for limitations, this solution only falls short in its inability to provide users with the option to define false routes simulating movement (e.g., by car or on foot), being limited to *static* locations.

Concerning the blocking of access to location data by third-party libraries, this implementation is practically constrained, as it necessitates a manual analysis of applications to determine which third-party libraries they employ. Subsequently, this information is stored on a remote server accessed each time the user installs a new application on their device.

Estrela [77] proposes a new approach to Android security by introspection. This solution, named *Sobek*, comprises an Android user application responsible for controlling the instrumentation behaviour. Additionally, it includes a *backend service* facilitating preference synchronisation across multiple devices, while also granting access to the *Sobek Instrumentation Tool* , which receives regular applications and applies instrumentation to them.

Figure 3.9 presents an overview of the interconnections between these components. The *Sobek Instrumentation Tool* is responsible for applying code modifications, while the *Sobek Manager* oversees the behaviour of instrumented applications. Additionally, the *backend* serves as an aggregation point for all server-side components, facilitating the delivery of instrumented applications, synchronisation of settings, and management of logs.

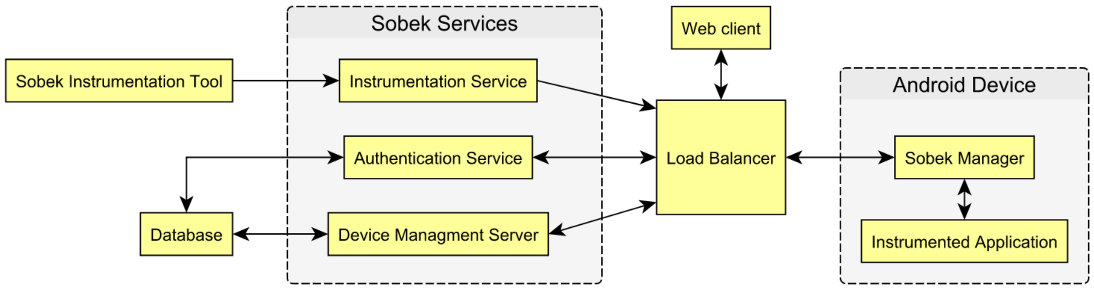


FIGURE 3.9: Sobek - System overview
Source: Adapted from [77]

This *unrooted* solution allows the user to redefine permissions granted to a specific application, tailoring them to their privacy needs. In addition to this functionality, it also enables the user to create false personas and provide fictitious locations to applications. However, there are some drawbacks to consider. Firstly, this solution requires the modification (instrumentation) of all applications the user wishes to control, which may hinder the user from receiving updates for the instrumented applications. As a result, the user must repeat the instrumentation process whenever they desire to update an application.

Regarding the provision of fictitious location data, this solution lacks the capability to provide simulated routes to mimic potential user movements (e.g., car or pedestrian routes). Furthermore, the instrumentation process may be challenging for an average user to comprehend, and the logs generated during the instrumentation process may also be difficult for them to understand.

Sobek relies on a *remote server* to perform the instrumentation and synchronisation of logs and preferences, causing this functionality to be intermittently available for users with limited mobile data subscriptions.

3.3 Theoretical Approaches

Theoretical approaches to privacy concerns of users regarding third-party access to their location data are those that have not been subjected to any implementation on mobile platforms nor tested in real-world scenarios. The vast majority of these mechanisms focus their approach on the tracking threat by obfuscating the precise user location, only revealing the characteristics of their mobility or a geographic region for their location (e.g., an area at the city level).

Considering the myriad of theoretical solutions that have been proposed in this domain in recent years, it would be impractical and beyond the scope of this academic work to provide a comprehensive presentation of all of them. Therefore, based on the aforementioned, we present a succinct overview of those proposals that have proven to be the most relevant and influential in shaping our own solution.

Krumm [78] proposes a method for generating realistic false paths to preserve location privacy. The main objective of this work is to enhance privacy, and the approach's effectiveness is assessed by an attacker's ability to distinguish false paths from real ones. The authors generate false paths by abstracting probabilistic models from real trajectories and utilising these probabilities to generate random start and end points, random routes, random speeds, and random GPS noise.

Palanisamy and Liu [79] proposes *MobiMix*, a *road network-based mix-zone* framework designed to safeguard the location privacy of mobile users while travelling on road networks.

The core idea in *MobiMix* is to disrupt the continuity of location exposure by employing *mix-zones*, where user movements cannot be traced by any applications. The authors contend that the construction and placement of effective *mix-zones* demand careful consideration of various factors, including the geometry of the zones, the statistical behaviour of the user population, spatial constraints on user movement patterns, and the temporal and spatial resolution of location exposure.

Furthermore, the researchers develop a suite of *road network mix-zone* construction methods aimed at achieving a higher level of attack resilience and ensuring a specified lower bound on the level of anonymity.

Meyerowitz and Roy Choudhury [80] introduces *CacheCloak*, a real-time location data anonymization system. *CacheCloak* operates as an intermediary server between users and LBS. When a user requests *location-centric data*, such as nearby restaurants based on their current location, the *CacheCloak* server responds by either providing cached data or fetching new data from the LBS.

Notably, instead of requesting data for a single GPS coordinate, *CacheCloak* seeks new data along an entire predicted path. This prediction extends until it intersects with other previously predicted paths. As a result, the LBS only receives requests from a series of interweaving paths, effectively preventing accurate tracking of any individual user.

Hara et al. [81] propose a privacy preservation method predicated on the utilisation of dummy entities with the aim of anonymizing the user's geographic location in a real environment. The technique entails the generation of said dummy entities in proximity to the user, a process which takes into consideration the actual geographical data.

Furthermore, the method imbues these dummy entities with simulated movement patterns to confer upon them a natural disposition during consecutive usage of the LBS. The authors premise rests upon the assumption that the user's mobile device proactively pre-fetches cartographic data pertaining to the locale in which the user is situated. Additionally, in this method, a user and dummies share the user's own registration ID on an LBS, which is beneficial both for the LBS provider and the user.

Upon the user's initiation of an LBS service request, the proposed methodology engenders the creation of dummy entities encompassing the user's position, adopting a *grid-like* configuration to meet the stipulated criteria for an anonymous spatial region.

During subsequent instances of service utilisation, wherein the user communicates their location data to the LBS, the method orchestrates the emulation of the dummy entities movements. This emulation is predicated upon the user's geographic coordinates and relevant spatial information, thereby ensuring compliance with the anonymous spatial domain requirements and determining the subsequent positions of the dummy entities. Additionally, the method strategically facilitates an interaction between the user and the dummy entities, serving to mitigate the traceability of the user.

Bindschaedler and Shokri [82] propose a methodology for generating *synthetic* yet semantically authentic privacy-preserving location traces. In this approach, the authors propose two mobility metrics that gauge the authenticity of a *synthetic* location trace concerning both the geographical and semantic dimensions of human mobility. These metrics are employed to construct a probabilistic generative model, which produces *synthetic* yet credible traces based on these metrics.

The development of this generative model draws from a dataset of actual location traces as its foundation, necessitating privacy preservation within the model itself. Consequently, the authors design privacy tests aimed at governing and curbing the potential leakage of information about the foundational seed dataset. Subsequently, *state-of-the-art* location inference attacks are employed to assess the efficacy of the *synthetic* traces in upholding the privacy of LBS users.

Through empirical evaluation conducted on a real location traces dataset, the authors demonstrate that by employing this methodology, the attacker's likelihood of inaccurately estimating users true locations over time stands at 0.9972. In other words, the approach attains a high degree of privacy protection.

The proposed scheme rests upon the notion that the mobility patterns of distinct individuals exhibit semantic similarity, irrespective of the geographic locations they visit.

The shared attributes in human mobility patterns emerge from comparable lifestyles, encompassing movements between home, workplace, friends residences, preferred shops, recreational venues, and occasional new destinations. These mobility patterns possess a common structure, encapsulating the collective behaviour of the population at a macro level.

The authors model individual mobility within two dimensions: *geographic* and *semantic*. In addition to these shared mobility patterns reflecting urban movement, geographical features tend to be distinct to each individual (e.g., the geographically unique location referred to as *home* for each individual). Conversely, semantic attributes tend to be universal and indicative of overall human mobility tendencies (e.g., the presence of an overnight *home* for most individuals).

3.4 Summary

In this Chapter, we proceed with the study of similar solutions. In this analysis, we observed a predominance of solutions that require the use of *rooted* devices or, alternatively, a custom ROM.

To facilitate a visual comparison of the differences between the various applications, Table 3.1 summarises the main functionalities offered by each of them.

Features	Refine-Droid Mr. Hide Dr. Android	LP-Guardian	MockDroid	TaintDroid	LP-Doctor	AppFence	Koi	MoveWithMe	PmP	Sobek	Our Solution
Rooted device or custom ROM	–	●	●	●	–	●	–	●	●	–	●
Need APK modification	●	–	–	–	–	–	●	–	–	●	–
Control of location data access	●	●	●	●	–	●	–	●	●	●	●
Graphic user interface	–	●	●	●	●	●	–	●	●	●	●
User-generated fake location	–	–	–	–	–	–	–	–	–	–	●
Fake static location	–	●	–	–	●	●	–	●	●	●	●
Simulated route	–	●	–	–	–	●	–	●	–	–	●

● feature present
– feature absent

TABLE 3.1: Similar implementations comparison

Solutions that do not require acquiring *root* privileges or the use of custom ROM tend to be less versatile and robust in protecting user privacy regarding access to their location data.

Although in fewer numbers, some *non-rooted* solutions attempt to match the level of protection and versatility of *rooted* versions. However, to achieve this goal, they necessitate the implementation of highly performance-penalising mechanisms, significantly affecting the user experience. In other cases, modification of APK of applications to be installed on the device is required, which is a complex process and not always within the reach of most users.

The choice of a *rooted* solution was guided by the greater effectiveness and versatility that this type of approach allows to offer to users, as well as the need to enable the creation of realistic fake paths, as suggested in the proposal made by Krumm [78].

Chapter 4

System Design

During the course of the literature review, conducted in the preceding Chapter 3, a comprehensive exploration of diverse approaches concerning user control over access to their location data by applications installed on their device was undertaken.

The analysis conducted on the various existing solutions has revealed that the issue of controlling access to user location data can be addressed through a wide range of different strategies. However, regardless of the multifaceted approaches taken to tackle this problem, the examined solutions unanimously recognise location data as a key element that significantly impacts user privacy.

Among the myriad solutions examined, a conspicuous absence of mechanisms allowing *dynamic* falsification of location, based on user-defined route creation, is evident.

Drawing from the insights gleaned from existing solutions, our proposition, as articulated in Section 1.2, aims to empower the user with the capacity to dynamically establish routes, whether on foot or by automobile. These simulated routes can then be employed as *mock location data* for applications residing on their device. In addition to this facet, our approach also encompasses the provision for users to designate *static* locations to be presented instead of their actual coordinates.

4.1 Threat Model

Threat modeling constitutes a foundational aspect in crafting a solution encompassing user privacy protection. In the conceptualisation of our proposed solution, we assume an honest yet curious and passive adversary, whose aim resides in inferring additional insights about the user based on accrued location data. We hold that applications serve as the sole conduit through which this adversary may access user location, leveraging the location API provided by the Android platform.

The adversary will not attempt to subvert the system or bypass any privacy controls that may be in place. It's important to emphasise that the security issues of the Android platform are orthogonal to our proposal, meaning our objective isn't to implement a solution that prevents an adversary from circumventing the security mechanisms of the operating system, in order to compromise our solution and get access to the real user location. The existence of such a solution would, in fact, strengthen our own solution.

The adversary will access the user's location data during the application's operation, and this access to information results in the following types of threats to user privacy:

Tracking Threat: With continuous access to location data, the adversary can real-time track the user, as well as identify the user's mobility patterns (frequent routes), and predict their future location with high precision by leveraging the typical consistency of people's mobility patterns [83].

Identification Threat: Sporadic access to location data may enable the adversary to pinpoint locations frequently visited by the user, such as their residence and workplace, subsequently using these as pseudo-identifiers to deduce the user's identity from anonymous location traces [27] [84].

Profile Threat: Mapping the user's mobility may exclude revealing identity associated locations and instead encompass places that the adversary can use to construct a profile. Such locations could include healthcare or educational institutions, religious sites, cultural event venues, and more.

We assume that the applications constitute a single recipient of location data and that all of these originate from the same developer or possess the same signature. We also assume that the underlying operating system is trustworthy, as implementing such a solution without this trust is practically unfeasible. Lastly, we assume that the user trusts their device and the underlying operating system to store and manage all of their personal information.

4.2 Architecture

Our solution encompasses three fundamental functionalities. It allows for the creation of simulated routes or fake *static* location points based on user preferences. It enables the user to select, from the applications installed on their device, those that will receive *mock location data*, and provides this *mock location data* to the applications chosen by the user to receive this type of information. In the Figure 4.1, we can visually observe the components that constitute our solution's architecture, as well as the interactions among them.

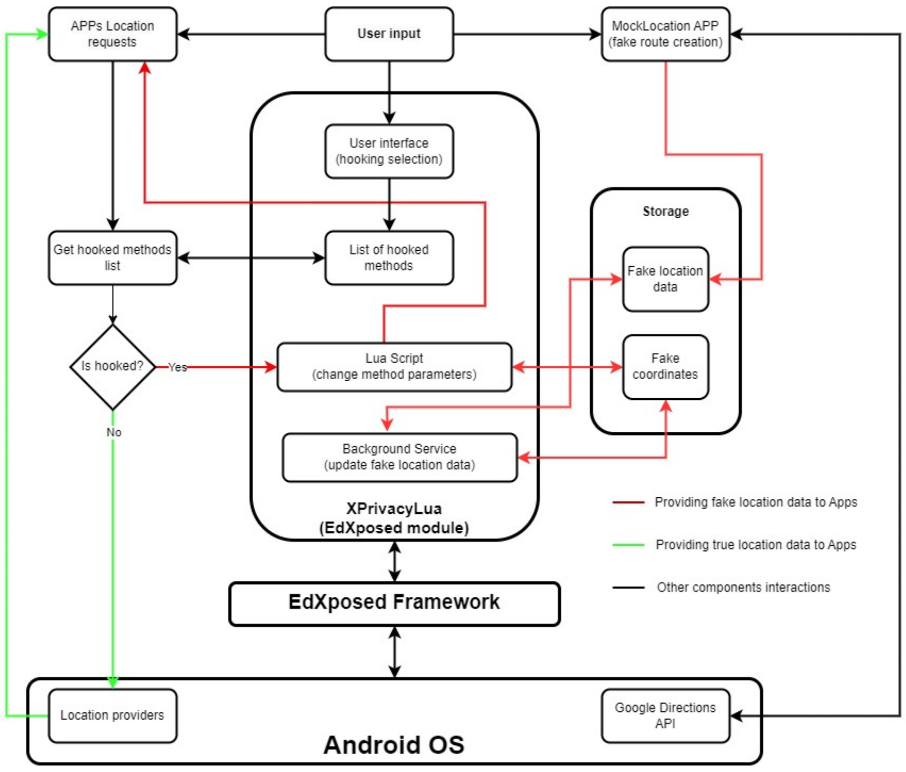

APPs Location requests
User input
MockLocation APP (fake route creation)
User interface (hooking selection)
Storage
Get hooked methods list
List of hooked methods
Fake location data
Is hooked?
Yes
Lua Script (change method parameters)
Fake coordinates
No
Background Service (update fake location data)
XPrivacyLua (EdXposed module)
Providing fake location data to Apps
Providing true location data to Apps
Other components interactions
EdXposed Framework
Location providers
Google Directions API
Android OS

FIGURE 4.1: System architecture and interactions of its components

The architecture of our solution is built upon three essential components: the *MockLocation* application, the *XPrivacyLua* module, and the *EdXposed* framework. The *MockLocation* application can be launched by the user at any time, with its primary functionality being to allow the user to create fictitious location data whenever they wish. This location data is accessed by the *XPrivacyLua* module, which has the functionality of enabling the user to intercept applications access to their real location data, providing them with fictitious location data without compromising the application's functionality.

The *EdXposed* framework provides the necessary API for the *XPrivacyLua* module to *hook* into the methods used by applications to access various location providers offered by the Android platform. In the subsequent sections of this chapter, we will describe the functionalities of the components within our solution and how they work together to achieve the objectives we outlined in Section 1.2.2.

4.2.1 MockLocation Application

In order to provide users with complete control over the creation of *mock location data* to be supplied to their selected applications, we have developed the *MockLocation* application. This application allows users to choose from various types of *mock* locations, which can be either related to fake fixed points on the map or simulated routes based on the user's choice of route way-points.

To generate false location fixed points, the user has two options. They can either drop a *pin* on the map at any location of their choice or choose to provide their current location, which will be converted into a fixed coordinate. The latter option is intended to allow the user to set their current position as *static*, so they can keep moving without it being noticeable to the applications selected to receive the fake location data.

The *MockLocation* application also allows for the simulation of routes, which can be either for driving or walking. To create a route, the user first needs to choose whether they want to create a walking or driving route. After that, the user selects two way-points for the route by placing two *pins* on the map at any location of their choice. Once the *pins* are placed at the chosen locations, the user clicks on the route creation button, and the application generates the route. For simplicity, the route begins and ends at the user's actual location, passing through the two fixed points selected by the user.

After obtaining the user's choices for *mock locations*, the *MockLocation* application sends a request to the *Google Directions* API to acquire coordinates for the respective locations or routes. Once the coordinates for the user-defined *mock locations* are received, the *Mock-Location* application stores this information in device storage, as depicted in Figure 4.1 as `Storage`.

The data placed in the device storage by the *MockLocation* application is intended to be accessed by the *XPrivacyLua* module, as detailed in the upcoming Section 4.2.2. In addition to the coordinates obtained from the *Google Directions* API, the *MockLocation* application stores information about the type of fictional location chosen by the user (*static*/*dynamic* and walking/driving) on the device storage.

4.2.2 XPrivacyLua Module

The *XPrivacyLua* module, developed by Marcel Bokhorst @ M66B [85], serves as the foundation for our work. Since this software is open source and distributed under the GNU GENERAL PUBLIC LICENSE, Version 3, 29 June 2007, we chose to modify its source code rather than developing a similar solution from the ground up.This decision was made to optimise resource usage while achieving our objectives.

The most significant modification made to the *XPrivacyLua* module was the addition of *dynamic* fictitious location data generation capability. This enhancement allows users to simulate walking or driving routes as needed.

As depicted in Figure 4.1, the modified version of the *XPrivacyLua* module is responsible for allowing users to select, from among the applications installed on their device, those that will receive simulated location data. In addition to this functionality, this module is also accountable for updating the location data through the execution of a *background service* (this task is crucial for dynamically providing location data). Furthermore, it delivers the fictitious location data by *hooking* into the methods used by the applications, previously selected by the user, to obtain the location data.

Considering that applications have various options for obtaining device location data, the *XPrivacyLua* module *hooks* into the methods used to access the following location providers:

Fused Location Provider: This method is the most recommended for accessing device location data. It combines multiple sources like GPS, WiFi, and cellular data to provide accurate and efficient location updates.

GPS Provider: Apps can request location updates from the GPS provider using the `LocationManager` class. This method provides high-precision location data but may consume more battery.

Network Provider: This method uses network-based location data, such as WiFi and cellular networks, to determine the user's location. It's less accurate than GPS but consumes less power.

Passive Provider: This provider listens for location updates generated by other apps or services on the device. It doesn't actively request updates but can be useful in scenarios where you want to passively listen to location changes.

When the user intends to provide a fictitious location dynamically (simulating a walking or driving route), the *XPrivacyLua* module will process the route's coordinates, obtained by the *MockLocation* application, in order to simulate the movement of a real user. This process will be discussed in detail in Chapter 5.

4.2.3 EdXposed Framework

As mentioned in Section 2.3, *EdXposed* is an extended version of the *Xposed* framework. This framework can be viewed as the API used by the *XPrivacyLua* module to perform method *hooking* on the applications selected by the user. To better grasp the role of the *EdXposed* framework, we need to briefly delve into the Android startup process and how applications are initialised.

As part of the startup process, the Android system needs to create virtual machines and the runtime environment for all system services that are compiled into the Dalvik Executable (DEX) Intermediate Language (IL). Additionally, it must offer a means to start and initialize fresh runtime environments for new applications as they commence. To address this requirement, Android employs an ingenious program named *Zygote*.

Zygote serves a similar role in Android as *init* does in Linux: it acts as the parent of all applications. During the system's startup, *init* initiates the launching of *Zygote*.

Zygote proceeds to self-initialize by pre-loading the complete Android framework. Unlike desktop *Java*, it doesn't employ lazy loading of libraries; instead, it loads all of them during the system's startup phase. Once fully initialised, it enters a continuous loop, waiting for connections to a *socket* [86].

When the system needs to launch a new application, it establishes a connection with the *Zygote socket* and sends a small packet containing information about the application to be started. *Zygote* replicates itself, creating a new kernel-level process. Figure 4.2 showcases the memory arrangement of a recently cloned application originating from *Zygote*. This novel application possesses its individual page table, with the majority of this new page table essentially mirroring that of *Zygote*.

Most modern Android devices start two *zygotes* — one for 32-bit applications and one for 64-bit apps—and default to 64-bit version [86].

The application actually started as the *root* user with the highest priority by the *init* process is located at `/system/bin/app_process` [86]. This is where *EdXposed* comes into action. Upon installing the framework, an extended `app_process` executable is copied to

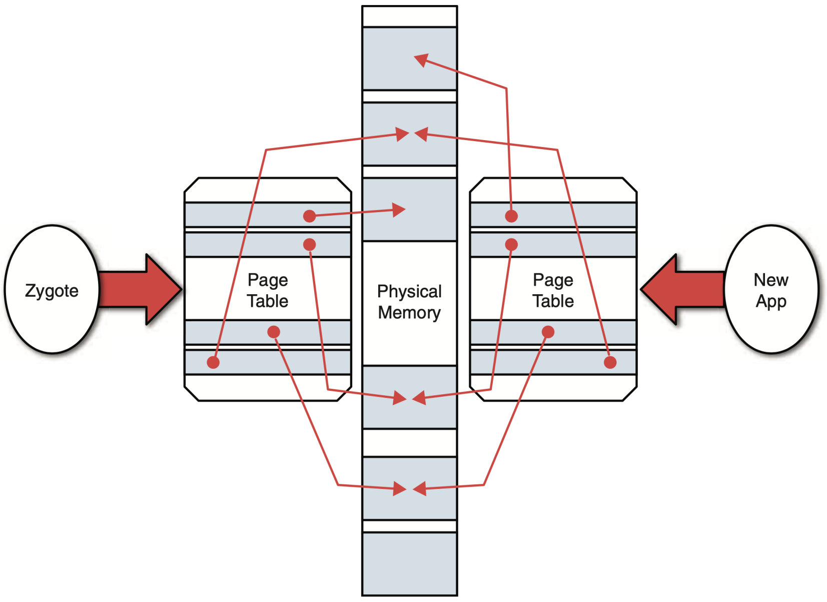


FIGURE 4.2: Zygote Clone
Source: Adapted from [86]

/system/bin. This extended startup process introduces an extra Java Archive (JAR) to the classpath and invokes methods from it at specific points. For example, right after the Virtual Machine (VM) has been established, even before the main method of *Zygote* has been called. Within that method, we are part of *Zygote* and can act in its context [87].

What truly empowers *EdXposed* is its capability to *hook* method calls. When modifying an APK by decompiling, we can directly insert or alter commands as desired. Nevertheless, recompilation and signing of the APK are necessary afterward, and only the entire package can be distributed.

With the *hooks* available through *EdXposed*, altering code within methods isn't feasible (as defining the precise changes and their locations would be challenging). Instead, we can inject our custom code before and after methods, which represent the smallest unit in Java that can be distinctly addressed.

XposedBridge[1] [88] incorporates a private native method called hookMethodNative. This method is also implemented in the extended app_process. It transforms the method type to *native* and associates the method's implementation with its proprietary native, generic method. Consequently, whenever the *hooked* method is invoked, the generic method is activated without the caller's awareness.

[1]The *XposedBridge* is an *EdXposed* class that encompasses the majority of *Xposed's* core logic, including initialisation and callbacks employed by the native side. It also incorporates functions for introducing new *hooks*.

Within this process, the `handleHookedMethod` method in *XposedBridge* is invoked. It passes the arguments, the `this` reference, and more to the method call. Subsequently, this method is responsible for triggering registered callbacks meant for this method call. These callbacks can modify call arguments, instance/*static* variables, invoke other methods, manipulate results, or skip any of these actions.

4.3 Summary

In summary, our proposal is based on two fundamental objectives. The first one consists of providing the user with an easy and intuitive way to select, from the applications installed on their device, those that can access their real location data. The second objective aims to allow the user to customise the type of fictitious location data that will be supplied to the applications they have chosen to receive false location information.

As described throughout this Chapter, the architecture of this solution involves the utilisation of three fundamental components, with each assigned a specific task in the process of concealing the user's actual location. However the interaction between them is crucial for the final outcome.

The creation of our solution involved implementing the *MockLocation* application, which allows for the generation of fake location data and modifying the code of the *XPrivacyLua* module, enabling the use of this software to dynamically provide simulated route location data.

In addition to developing the *MockLocation* application and making modifications to the *XPrivacyLua* module's code, we also established a setup to support the operation of our solution, as described in Section 5.1. This setup primarily focuses on the device *rooting* process and the installation of various frameworks that enhance the functionality of our solution.

The following Chapter 5, in which we will delve into the implementation of our solution, will provide a more detailed description of how the components shaping our architecture function.

In Chapter 6, where we will present the results of our solution's testing, we will offer a more visual description of how the various components of our solution operate and how their interaction enables us to achieve the goals outlined in the Section 1.2.

Chapter 5

Implementation

In Section 4.2, we presented the components comprising our solution. This chapter delves into the complete implementation process, along with the challenges that emerged as we advanced in constructing our solution.

In order to provide a holistic perspective on our implementation, this chapter begins by providing a description of the supportive setup for the implementation. It subsequently progresses to elaborate on the remaining stages of development, which encompassed the implementation and adaptation of the components that underpin our solution.

5.1 Initial Setup

To develop our solution, it was necessary to create an initial setup, bringing together a set of tools and supporting components for implementing the functionalities described in Section 1.2.2. In this section, we describe the steps involved in establishing the setup to support the implementation of our solution.

5.1.1 Rooting the Device

With the technological advancements characterising our current era, Android devices have garnered significant interest that extends beyond their mere utilisation. Nowadays, we observe an increasing number of users seeking to modify and personalise these systems. In this context, the concept of *rooting* emerges, encompassing a process through which elevated administrative privileges can be obtained over the Android OS.

The *rooting* process enables users to access areas of the system that would typically remain inaccessible. Despite Android being regarded as an *open-source* OS, certain software configurations and hardware resources are only accessible with *root* level access.

The *root* access facilitates a range of operations, spanning from the installation of custom applications to the optimisation of device performance. However, it is worth noting that the *rooting* process is not devoid of risks, necessitating users to exercise a more judicious and cautious management of their devices from a security standpoint.

Considering our choice to develop a solution that requires *root* access to the device, this section outlines the process of *rooting* the device, including software-level modifications and the tools used to accomplish this task.

5.1.2 The adb Tool

The *Android Debug Bridge (adb)* is a versatile *command-line* tool, included in the *Android SDK Platform Tools* package, that enables communication with a device [89]. Executing the `adb` command allows for a variety of actions on the device, including debugging and application installation. *The adb* tool provides access to a *Unix shell* that can be used to execute various instructions on the device. It is a *client-server* program composed of three components:

A *client* responsible for sending commands.

A *daemon* (*adbd*) that executes commands on the device, functioning as a *background process* on each device.

A *server* that manages communication between the client and the *daemon*.

5.1.3 The Backup Process

Considering that the *rooting* process carries some risks, it becomes imperative to perform a backup of all its original partitions before making any changes to the device.

Prior to commencing the *rooting* process, we performed an examination of the device's features by executing the following commands through the adb tool:

`$adb shell getprop ro.build.version.release` - To get the Android version release.

`$adb shell getprop ro.build.version.sdk` - To retrieve the Android SDK version.

`$getprop ro.secure` - To check if the phone is already rooted.

To perform the backup of all partitions in their original state, it's necessary to run a custom recovery software in remote mode (i.e. without installing the custom recovery software) and having *root* privileges. By doing so, we prevent modifying the recovery partition, allowing us to restore the device to its initial state in the future. To perform this task, we employed the Team Win Recovery Project (TWRP) recovery software, executing the command `$fastboot boot twrp.img`.

TWRP is the leading custom recovery for Android phones. A custom recovery is used for installing custom software on the device. This custom software can include smaller modifications like *rooting* the device or even replacing the firmware of the device with a completely custom ROM [90].

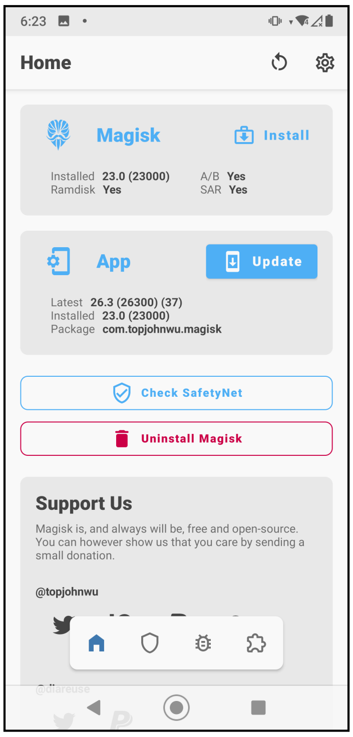
Running TWRP in remote mode allows for the installation of the *Magisk* application on the device. The *Magisk* application enables the acquisition of *root* privileges on the device and, consequently, the backup of all partitions using the *adb* tool.

5.1.4 Magisk - Systemless Root

Systemless rooting is considered a more recent workaround for advanced users who wish to utilise services that check for *rooting* without needing to *unroot* their devices [91]. This process involves modifying the device's boot partition instead of altering the system partition, by injecting the necessary files into the boot partition to grant *root* privileges to third-party apps, install custom *ROM*, or perform other advanced tasks without modifying the system partition.

One of the main advantages of *systemless rooting* is that it allows you to continue receiving OTA updates from the device manufacturer while maintaining *root* privileges, as long as the update doesn't overwrite the device's boot partition. Due to its non-alteration of the system partition, this *rooting* method is also less risky and easier to revert to a *non-root* state.

Magisk is an open-source *root* solution compatible with a wide range of Android devices. This tool enables the installation of modules that make changes to the Android operating system without affecting the system partition. In broad strokes, its operation involves installing a modified boot image that intercepts calls to the system partition and redirects them to a virtual partition created in the data partition. This virtual partition contains the modified files and modules installed by *Magisk*. Figure 5.1 illustrates the layout of this application after being installed on the device.



Installed 23.0 (23000)
Ramdisk Yes
A/B Yes
SAR Yes

Latest 26.3 (26300) (37)
Installed 23.0 (23000)
Package com.topjohnwu.magisk

Magisk is, and always will be, free and open-source. You can however show us that you care by sending a small donation.

@topjohnwu

FIGURE 5.1: Home screen of the Magisk App

The authors of *Magisk* offer various types of modules, including solutions that enable evading *root* detection methods implemented by Google, such as *SafetyNet* [92] and *Google Play Protect*, as well as continuing to enjoy OTA updates [93].

5.1.5 Installation of the EdXposed Framework

After obtaining *root* privileges on the device and having installed the *Magisk* application, we are in a position to install the *EdXposed* framework, which is one of the three fundamental components of our solution.

After Android 8.0 (Oreo), the developers of the *Xposed* framework ceased to produce new versions of this software. *EdXposed* provides an alternative to the *Xposed* framework, and it is compatible with Android 8.0, Android 9.0, and Android 10.0. The functioning of this framework is based on the coordination among the following components:

YAHFA: *YAHFA* is a *hook* framework for Android ART. It provides an efficient way for Java method *hooking* or replacement [61].

Riru: *Riru* inject into *zygote* in order to allow modules to run their codes in apps or the *system server* [94].

XposedBridge: Is the original *Xposed* framework API [95].

Dexmaker and dalvikdx: Used to dynamiclly generate *YAHFA hooker* classes [96] [97].

SandHook: ART *hooking* framework for *SandHook* variant [98].

Dobby: Used for inline *hooking* [99].

Magisk: *Magisk* is a suite of open-source software for customising Android, supporting devices running Android 6.0 and higher. The main functionalities [100] of this module are as follows:

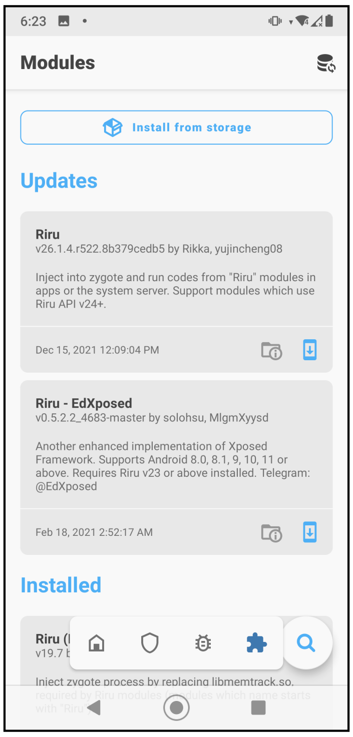
> ***MagiskSU***: Provides *root* access for applications.
>
> ***Magisk Modules***: Modify read-only partitions by installing modules.
>
> ***MagiskBoot***: The most complete tool for unpacking and repacking Android boot images.
>
> ***Zygisk***: Run code in every Android applications processes.

In order to have the *EdXposed* framework installed on our device, after installing *Magisk* as described in Section 5.1.4, we use this application to install the *Riru* module and finally install the *EdXposed* framework. Figure 5.2 shows the *Riru Core* module and the *Riru - EdXPosed* module installed on the device.

The installation of the *EdXposed* framework is only complete after installing the *EdXposed Manager*. This application (see Figure 5.3) for Android is used to manage the *EdXposed* framework on *rooted* devices using Magisk.

5.1.6 Installing the XPrivacyLua Module in EdXposed

In order to use the *XPrivacyLua* module [85] to manage the access of location data for other applications installed on the system, it needs to be installed within the *EdXposed* framework (see Figure 5.3b).


6:23

Modules

Install from storage

Updates

Riru
v26.1.4.r522.8b379cedb5 by Rikka, yujincheng08

Inject into zygote and run codes from "Riru" modules in apps or the system server. Support modules which use Riru API v24+.

Dec 15, 2021 12:09:04 PM

Riru - EdXposed
v0.5.2.2_4683-master by solohsu, MlgmXyysd

Another enhanced implementation of Xposed Framework. Supports Android 8.0, 8.1, 9, 10, 11 or above. Requires Riru v23 or above installed. Telegram: @EdXposed

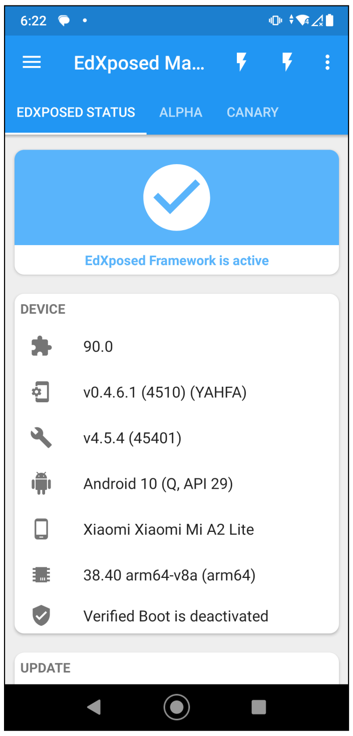
Feb 18, 2021 2:52:17 AM

Installed

Riru (
v19.7 b

Inject zygote process by replacing libmemtrack.so.

FIGURE 5.2: Riru- Core and Riru - EdXposed installed in Magisk


6:22

EdXposed Ma...

EDXPOSED STATUS ALPHA CANARY

EdXposed Framework is active

DEVICE

90.0

v0.4.6.1 (4510) (YAHFA)

v4.5.4 (45401)

Android 10 (Q, API 29)

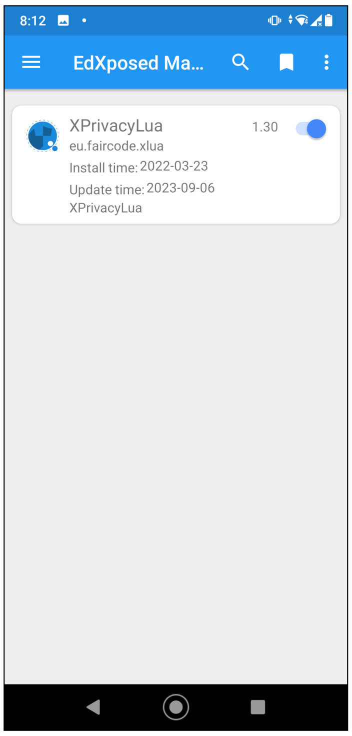
Xiaomi Xiaomi Mi A2 Lite

38.40 arm64-v8a (arm64)

Verified Boot is deactivated

UPDATE

(A) EdXposed - Main layout


8:12

EdXposed Ma...

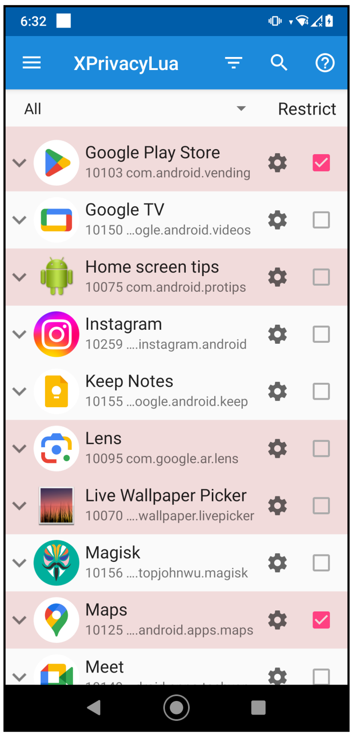
XPrivacyLua 1.30
eu.faircode.xlua
Install time: 2022-03-23
Update time: 2023-09-06
XPrivacyLua

(B) EdXposed - Modules installed

FIGURE 5.3: EdXposed Manager

The *XPrivacyLua* module, whose layout is displayed in Figure 5.4, primarily functions to empower the user to select the applications to which false location data, generated by the *MockLocation* application detailed in the following Section 5.2, will be provided.

The original version of the *XPrivacyLua* module does not allow for the *dynamic* provision of fake location data. As mentioned in Section 1, one of the intended functionalities for our solution was the ability for the user to generate false routes, either on foot or by car, in order to simulate a journey. To fulfil this objective, we conducted an analysis of the *XPrivacyLua* module's code, identifying the necessary changes to implement this feature. This process is described further in Section 5.3.


6:32

XPrivacyLua

All — Restrict

Google Play Store	10103 com.android.vending	✓
Google TV	10150 ...ogle.android.videos	☐
Home screen tips	10075 com.android.protips	☐
Instagram	10259 ....instagram.android	☐
Keep Notes	10155 ...oogle.android.keep	☐
Lens	10095 com.google.ar.lens	☐
Live Wallpaper Picker	10070 ....wallpaper.livepicker	☐
Magisk	10156 ....topjohnwu.magisk	☐
Maps	10125 ....android.apps.maps	✓
Meet		☐

FIGURE 5.4: XPrivacyLua module

5.2 Developing the MockLocation Application

The requirements for providing false location data can vary based on the device user, as well as the applications for which they intend to supply fake location data. Keeping this consideration in mind, we have developed an application named *MockLocation* (see Figure 5.5 and Figure 5.6), which enables the user to create four types of false locations.

The four types of simulated location generated by the *MockLocation* application, as shown in Figure 5.6, have the following characteristics:

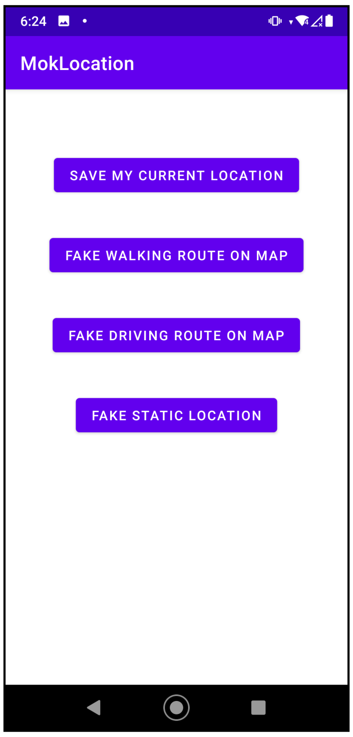
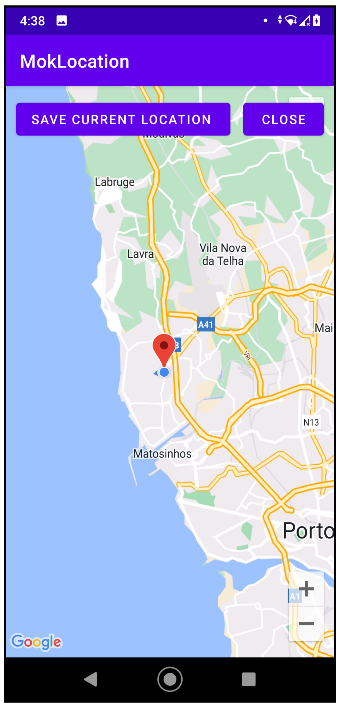


FIGURE 5.5: MockLocation main activity

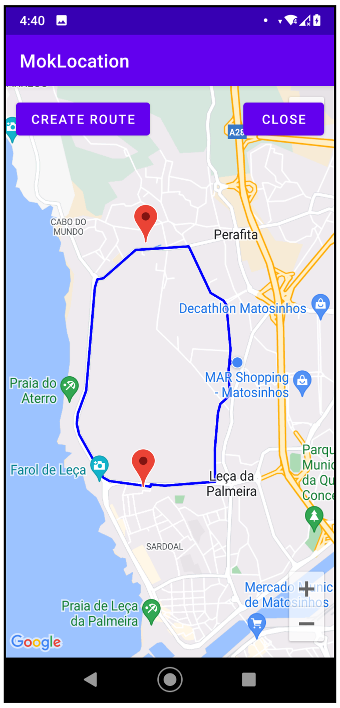
Setting the current location as permanent: When this option is selected, the application saves the coordinates of the user's current position in the device storage. This becomes the information provided to all selected applications for receiving fictitious location data.

Creating a walking route: Upon selecting this option, the user is directed to a map where they can choose two points for the route. For simplicity, all routes start and end at the user's current position, passing through the two chosen points. To obtain the route coordinates, the *MockLocation* application calls the *Google Directions* API, specifying the coordinates of the starting point, the endpoint, and any intermediate points of the route. In addition to these arguments, the desired mode of transportation is also specified in the call to the *Google Directions* API. After obtaining the coordinates of the route, the *MockLocation* application stores this information in the device's memory.

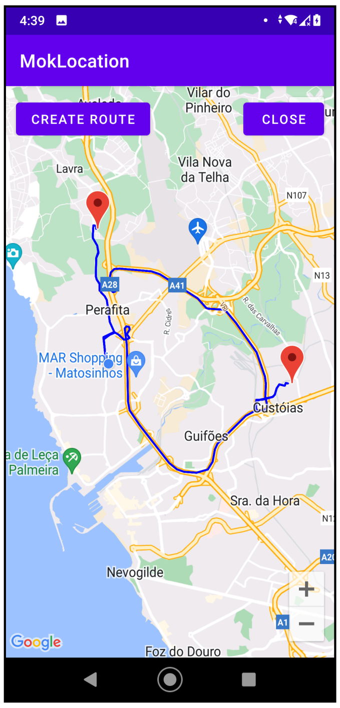
Creating a car route: This option is similar to the previous one, with the parameters passed to the *Google Directions* API corresponding to a car journey instead of walking, as in the previous option.



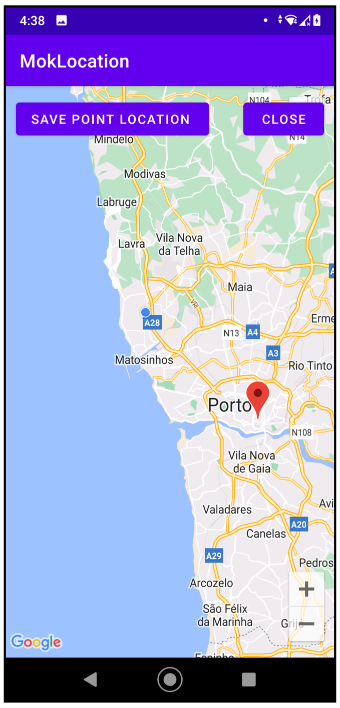
(A) Save current location



(B) Create walking route



(C) Create driving route



(D) Create static location

FIGURE 5.6: MockLocation - Working modes

Setting a static location point: In this option, the user can choose a fictional location of their choice on the map. This is done by placing a *pin* on the map and clicking a button to instruct the application to save this information. The application will store the coordinates of the user's chosen point in the device's memory.

In addition to the *user-defined fictional location coordinates*, the *MockLocation* application also saves a *code-word* in the device's memory, related to the chosen mode of fictional location. This information is crucial for the functioning of the *XPrivacyLua* module, as we will see in the next section.

The *MockLocation* application allows the user to change their fictitious location data at any time. This aspect aims to address the flexibility requirement mentioned in Section 1.2.2.

5.3 Changing the XPrivacyLua Module

As mentioned in Section 5.1.6, the original version of the *XPrivacyLua* module only allows providing a fixed coordinate as fictitious location data. This fixed coordinate is hardcoded in the script `location_createfromparcel.lua`, used by the *XPrivacyLua* module to supply fake location data to the user-selected applications by *hooking* into the methods they use to access the device's location data.

In order to enable the *XPrivacyLua* module to provide fake location data dynamically, it became necessary, among other changes, to allow the `location_createfromparcel.lua` script to update the fictitious coordinate values defined by the user through the *MockLocation* application every time it was executed. Therefore, to address this requirement, after conducting a thorough analysis of the entire *XPrivacyLua* module code, we made several modifications to the application's code, which are detailed throughout this section.

5.3.1 Changes to the Script `location_createfromparcel.lua`

As previously mentioned, in the original version of the *XPrivacyLua* module, the false location provided to applications results from inserting *hardcoded* fictitious *latitude* and *longitude* coordinate values into the Lua script (`location_createfromparcel.lua`). This script is responsible for rewriting the parameter values of methods used by applications to access location data, which are *hooked* using the *EdXposed* framework's API.

In order to allow the script `location_createfromparcel.lua` to update the fake location data with each execution, its code was modified to read the fictitious coordinates from the device's memory, rather than providing hardcoded coordinates as in the original version of the *XPrivacyLua* module.

The coordinates read by the script `location_createfromparcel.lua` from the device's memory are updated by the *XPrivacyLua* module through a *background service*, as described in the following section. The source code of the modified version of the Lua script `location_createfromparcel.lua` can be found in Appendix A.1.

5.3.2 Adding a Background Service

As highlighted in Section 5.1.6, the *XPrivacyLua* module enables users to choose, from among the applications installed on their device, those that will receive fictitious location data.

To allow the *XPrivacyLua* module to access the fictitious location data created by the *MockLocation* application, a *background service* has been integrated. This *background service* runs every time the *XPrivacyLua* module is launched by the user.

The *background service* operates seamlessly for the user, periodically verifying the selected type of fictitious location (refer to Section 5.2) and cyclically updating the fictitious location data (*latitude* and *longitude*) that will be provided to the user-selected applications requesting this kind of data.

The types of *mock* locations (*static* point, user's current location, car route, and walking route) are encoded using a *code-word* stored in the device's memory by the *MockLocation* application and read by the *background service* of the *XPrivacyLua* module.

For each chosen type of simulated fictitious location data, the *XPrivacyLua background service* launches a different *thread*. Depending on the type of fictitious data to be provided to applications (*static or dynamic*), it updates pairs of coordinates (*latitude* and *longitude*) used in *hooking* location data retrieval methods by the `location_createfromparcel.lua` script described in Section 5.3.1. A graphical depiction of the *background service* operation is presented in Figure 5.7.

As shown in Figure 5.7, the *background service* operates with the following logic:

1. The service is initiated whenever the user launches the *XPrivacyLua* module. This service continues to run until the *XPrivacyLua* module terminates its execution.

2. Once started, the *background service* reads the *code-word* stored in the device's memory by the *MockLocation* application.

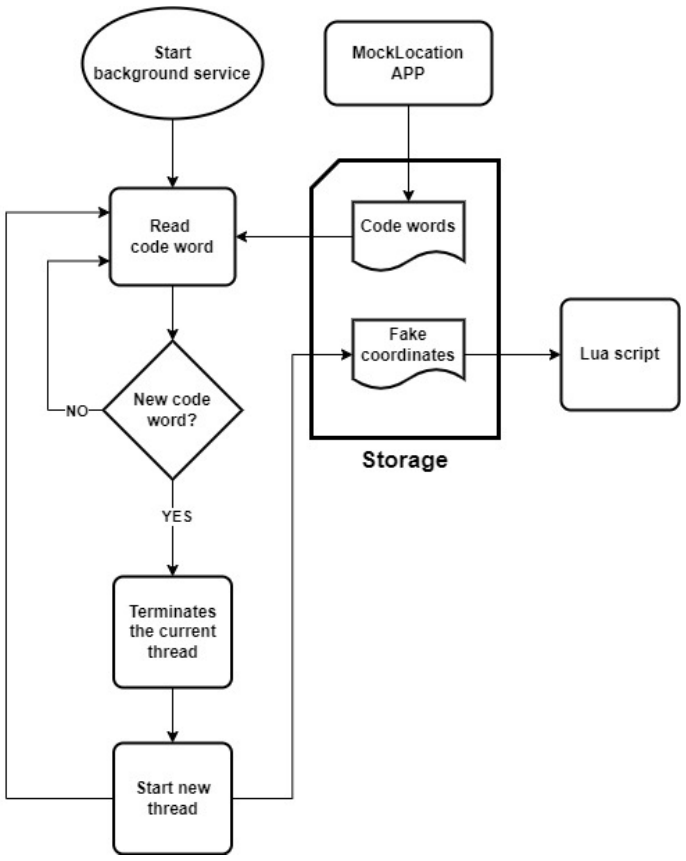


FIGURE 5.7: Operation of the background service added to the XPrivacyLua module

3. Based on the value of the *code-word*, the *background service* launches the corresponding *thread* responsible for processing the *mock location* data to be provided to the user-selected applications.

4. The *background service* then enters a *loop* to check the *code-word,* continuously. In this *loop*, whenever the *code-word,* is changed by the *MockLocation* application, the *background service* terminates the currently running *thread* and starts a new *thread* to process the *mock location* data related to the user's new choice.

Each new *thread* is responsible for the appropriate processing of *mock location data* to be provided to the user-selected applications. The specifics of how each *thread* processes the *mock location data* are described in the following sections. It is worth mentioning that the main difference in the operation of the *threads* responsible for handling the fictitious location data arises from the user's choice to provide either *static* or *dynamic* location data, as we will see next.

5.3.3 Managing Simulated Location Data with Threads

As mentioned in the previous section, the *background service* is responsible for managing various tasks, including how fake location data is provided to the user-selected applications. While the *background service* launches a *thread* for each of the four types of fake locations, the key distinction lies between *static* and *dynamic* location data.

5.3.3.1 Handling Fake Static Location

When the user chooses to provide a fixed point as *mock location data*, the *background service* of the *XPrivacyLua* module launches a specific *thread* to handle this type of *mock location data*.

The operation of the *thread* responsible for handling *static* location data is depicted in Figure 5.8 and consists of the following steps:

1. Upon being launched, the *thread* reads the *mock location data* stored in the device's memory by the *MockLocation* application.

2. Next, the *thread* writes the *mock location data* into the device's memory so that it can be read by the `location_createfromparcel.lua` script during the process of *hooking* into location access methods used by applications.

3. The *thread* enters a *loop* in which it checks if the static location data has been altered by the *MockLocation* application. Whenever the user changes the *static mock location data*, the *thread* also updates the data to be read by the `location_createfromparcel.lua` script.

This process repeats itself as long as the *background service* does not terminate the *thread* due to a change in the type of *mock location data* chosen by the user (e.g., the user selects a *mock location* with a simulated route for a car). The *Java* class code responsible for handling *static location* data can be found in Appendix A.3.

5.3.3.2 Handling Dynamic Fake Location

When the user intends to provide *dynamic fake location* data, simulating a route by walking or driving, it becomes necessary to process the coordinates of these routes obtained by the *MockLocation* application in the route simulation process described in Section 5.2.

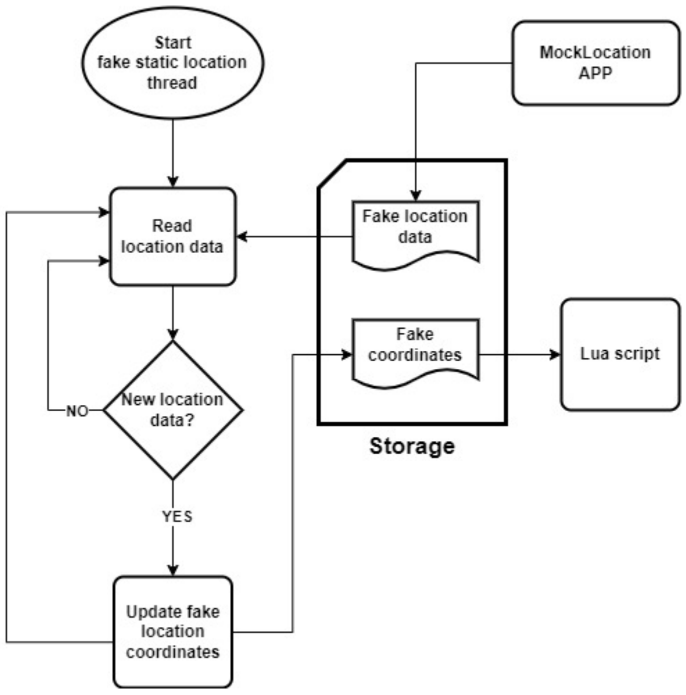


FIGURE 5.8: Execution flow of threads managing static fake location data

Before delving into a more comprehensive explanation of this procedure, Figure 5.9 illustrates the sequence of actions executed by the *threads* tasked with handling *dynamic* location data manipulation.

Similar to the processing of *static fake location* data, the data for simulated routes is processed by the *background service* through the execution of a specific *thread*.

The simulation of walking and driving routes is done in a similar manner, with the only difference being the speed of movement between the various points that make up the route. For the sake of simplicity, we will present only the simulation of a driving route. However, please note that all the Java code related to the *threads* responsible for handling *dynamic location* data is available in Appendix A.4.

The first action performed by the *thread*, handling *fake location data*, is to read the coordinates of the simulated route obtained from the *MockLocation* application through a call to the *Google Directions* API, as described in Section 5.2. Since directly using these coordinates to simulate a route would result in abrupt position variations, significantly different from the normal movement pattern of a user, we process the coordinates obtained from the *Google Directions* API to *smooth out* the position changes along the route, aiming to simulate a route that appears as close to reality as possible.

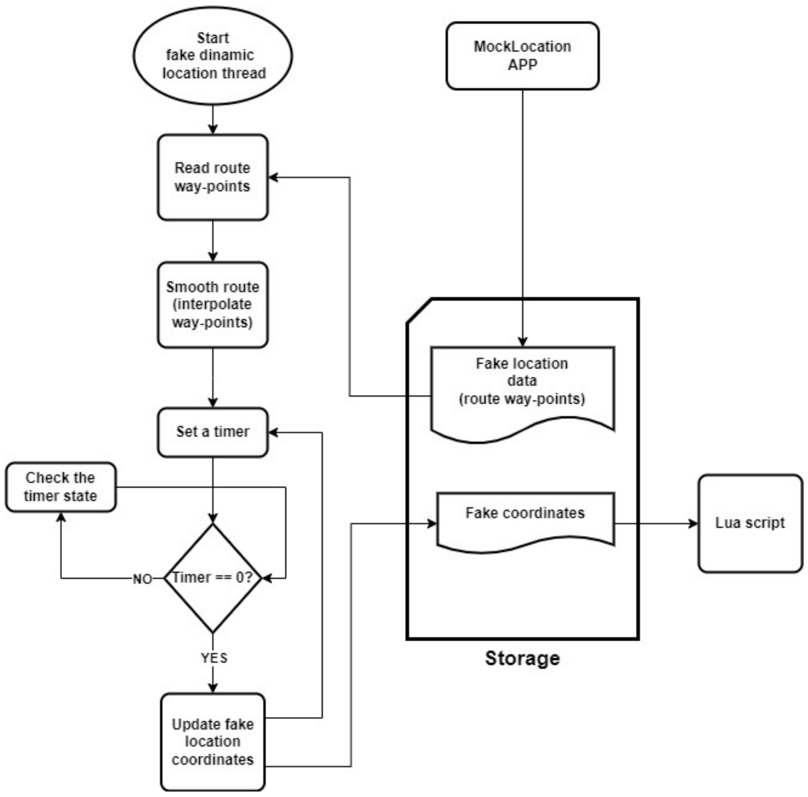


FIGURE 5.9: Execution flow of threads managing dynamic fake location data

Considering the impossibility of directly using the coordinates from the *Google Directions* API in route simulation, the execution of the *thread*, whose execution flow is shown in Figure 5.9, performs the following actions:

1. Upon initiation, it reads the coordinates of the route points provided by the *Google Directions* API.

2. It then defines the number of intermediate segments that will be calculated between each pair of consecutive coordinates provided by the *Google Directions* API. The number of intermediate segments always assumes a constant value, resulting from tests conducted with various types of routes.

3. After setting the number of intermediate segments between each point on the route provided by the *Google Directions* API, it defines the time interval that will be used to update the false coordinates to be provided to the user-selected applications for receiving this type of fake location data. Similar to the number of intermediate segments, the time interval for updating the fictitious location values also assumes a constant value, resulting from tests performed with various types of routes.

4. Next, it calculates the distance between two points on the route using the *Haversine* formula. The result of this operation is used to determine the length of the intermediate segments between each pair of coordinates provided by the *Google Directions* API.

5. After calculating the size of the intermediate segments between two points on the route provided by the *Google Directions* API, interpolation of the coordinates is performed, with intermediate coordinates being defined based on the length of the intermediate segments, resulting in a smoothing of the movement and giving a more realistic appearance to the simulated route.

Simulating a route involves movement with variable speeds. Since the coordinates provided by the *Google Directions* API have varying distances between them depending on the characteristics of each route, by setting a constant value for the number of intermediate segments between two points, as well as for the time interval for updating false coordinates, we obtain a simulated route with random variable speeds. This pattern of variable speed never repeats because it depends on the characteristics of each calculated route.

5.4 Summary

In summary, the implementation of our solution began with the process of *rooting* the Android device, which involved utilising various tools and applications that enable us to operate the device with administrative privileges (*root*). Among the set of tools used in the *rooting* process, it's worth highlighting the *Magisk* application, which enables a *systemless rooting* approach, and the *EdXposed* framework, allowing us to inject code into the *Zygote* process, from which all other processes are cloned.

Once the setup phase was overcome, the *MockLocation* application was developed to enable users to create both *static* and *dynamic* fictitious location data. Finally, leveraging an existing solution, an analysis was carried out on the code of the *XPrivacyLua* module, to which the necessary modifications were made to enable the simulation of walking and driving routes, as well as the selection of *static* user-defined location points.

Chapter 6

Evaluation

Considering that our solution consists of a *functional prototype*, its purpose is to demonstrate the concept that revolves around the ability to provide fictitious location data to applications installed on an Android device.

Given the complexity of the Android ecosystem and the various ways in which a particular application can infer the user's location, our expectation is that our solution may not yield the expected results in all applications whose functionality may be influenced by the user's location.

Considering that the applications provided by Google, in conjunction with the operating system, are not optional installations for users, the development of our solution focused on this type of application. This choice was made because these applications, being an integral part of the software that comes with the device, pose the highest privacy risk to users.

In order to assess the behaviour of other applications outside the realm of Google, we selected a set of the most popular applications on the *Google Play Store* with which we tested our solution.

Throughout this Chapter, we have presented the results of the tests conducted on our solution, starting with the pre-installed applications on the device, which are also widely used by Android users. Subsequently, we conducted tests with applications sourced from the *Google Play Store* that are not part of the Google ecosystem.

For the tests, we selected a set of applications whose functionality relies on the user's location and are concurrently commonly used among the Android device user community.

In order to test our solution in a real-world environment, we chose to acquire a device available in the market with the following specifications:

Device Model: Mi A2 Lite

Serial number: 9e45643d0305

Android version: 10 aka Android Q

Android SDK version: 29

Android security patch level: 1/07/2021

Baseband version: 953_GEN_PACK-1.240106.1.282152.1

Kernel version: 3.18.124-perf-g48df2b1

Build number: QKQ1.191002.002.V11.0.21.0.QDLMIXM

Wi-Fi MAC address: f4:60:e2:c6:14:cc

Bluetooth address: f4:60:e2:c6:14:cb

The device underwent the *rooting* process, and it was observed that it retained all the functionalities it had previously. All the tested applications were obtained from the Google Play Store and installed on the device without any modifications.

6.1 Google Maps

Google Maps is a globally used mapping application, regarded as the world's most popular navigation app [101] [102]. This application provides users with a wide range of navigation options, route planning, and location exploration.

In addition to navigation services, *Google Maps* also offers *real-time* data on road traffic, public transport schedules, and information about local businesses. Figure 6.1 illustrates the statistical data confirming the global popularity of the *Google Maps* application.

Considering the impossibility of demonstrating the operation of the *Google Maps* application with data from a simulated route in the *MockLocation* application, we opted, for the sake of simplicity, to demonstrate the operation of this application by providing *static* fictitious location data.

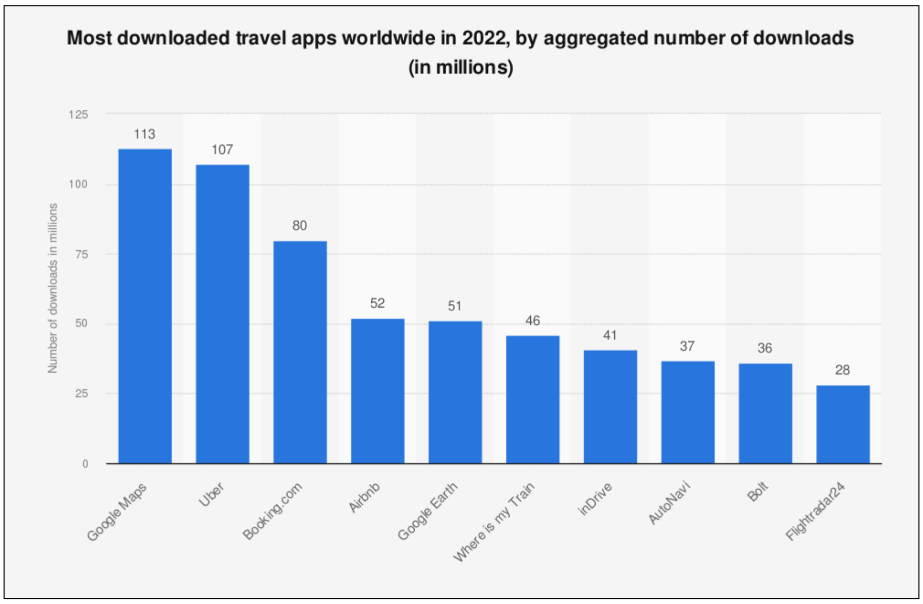


FIGURE 6.1: Most downloaded travel apps worldwide in 2022
Source: Adapted from [102]

After providing fictitious location data to the *Google Maps* application, we tested its LBS for the user's current location. The first test, as shown in Figure 6.2, illustrates the application using fictitious data to display the user's *current location*. In Figure 6.3, the test of the *remaining location-based services* of the *Google Maps* application is depicted.

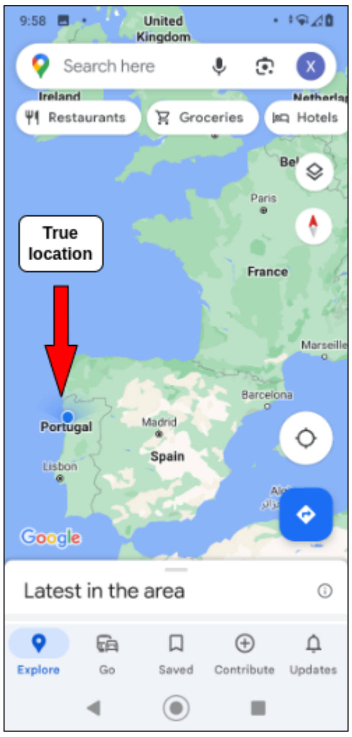
To make the application assume the fictitious data as the user's *current location*, the test described in Figure 6.2 was conducted, during which the following actions were performed:

1. The application was launched without any controls regarding device location data access, and it was observed that it accessed real location data (see Figure 6.2a).

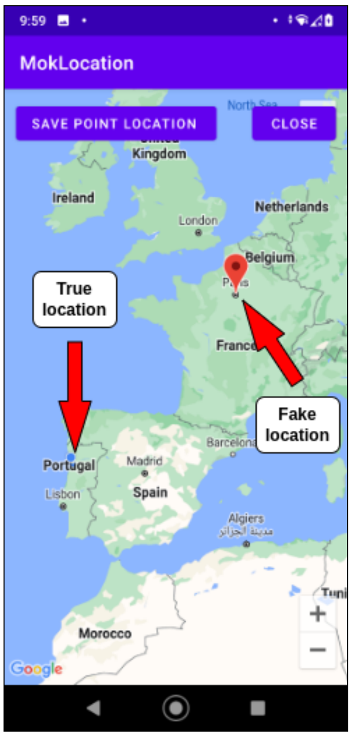
2. Next, the *MockLocation* application was used to set a fictitious location near the center of Paris (see Figure 6.2b).

3. After creating the fictitious location point, we launched the *XPrivacyLua* module and selected the *Google Maps* application to receive fictitious location data.

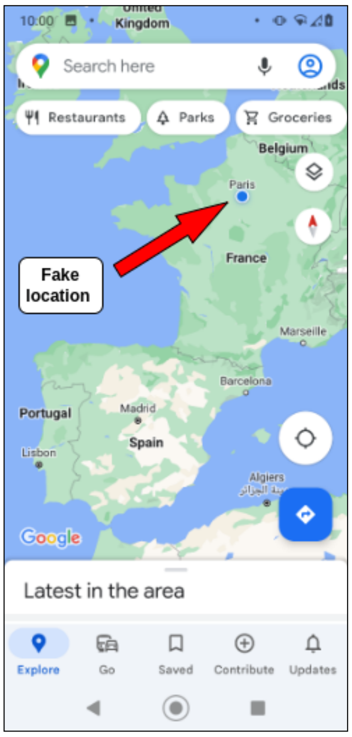
4. The *Google Maps* application was relaunched, and it was observed that it displayed the *current location* as the point corresponding to the fictitious data created by the *MockLocation* application (see Figure 6.2c).



(A) Google Maps showing the true current location



(B) Using the MockLocation app to choose a fake location point



(C) Google Maps showing the fake location

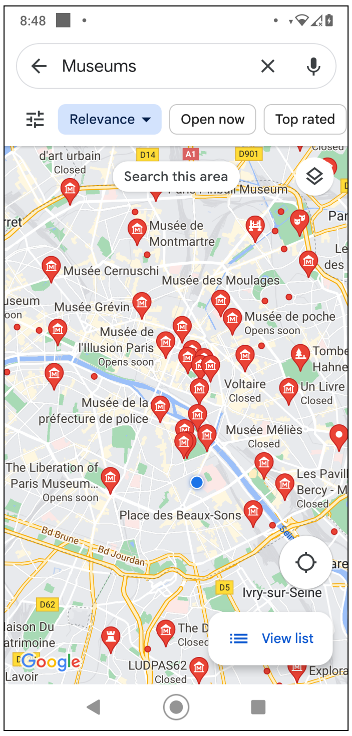
FIGURE 6.2: Testing the Google Maps application with a fictional location

As previously mentioned, to assess the functionality of our solution in the remaining services of the *Google Maps* application, we conducted the test described in Figure 6.3, in which the following services were tested:

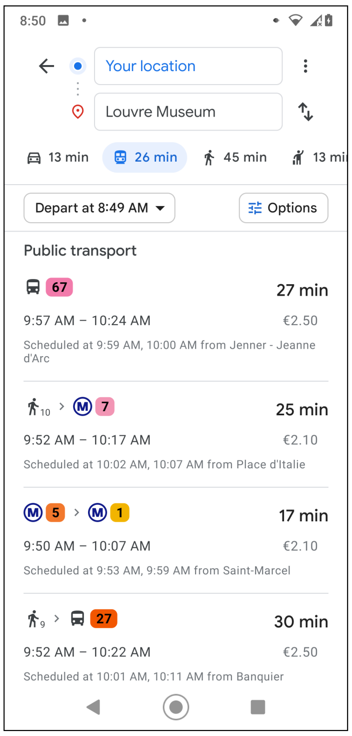
1. Searching for places of interest near the current location (food and drinks; things to do; shopping and services).
2. Public transport information.
3. Route calculation, starting from the fictitious location point.

As can be observed in Figure 6.3, the tested services of the *Google Maps* application continue to function normally with fictitious location data. Figure 6.3a displays the result of a search for museums near the user's *current location*. In Figure 6.3b, information about the available public transportation between the user's *current location* and the Louvre Museum is shown, and finally, in Figure 6.3c, the planning of a car route between the user's *current location* and the Louvre Museum is presented.

It's worth mentioning that similar results were obtained when we provided simulated routes to *Google Maps*. It was possible to observe a continuous change in the user's fake location as the fictitious coordinates were being provided.



(A) Google Maps searching for services (museums) with fake location


8:50
Your location
Louvre Museum
13 min 26 min 45 min 13 min

Depart at 8:49 AM Options

Public transport

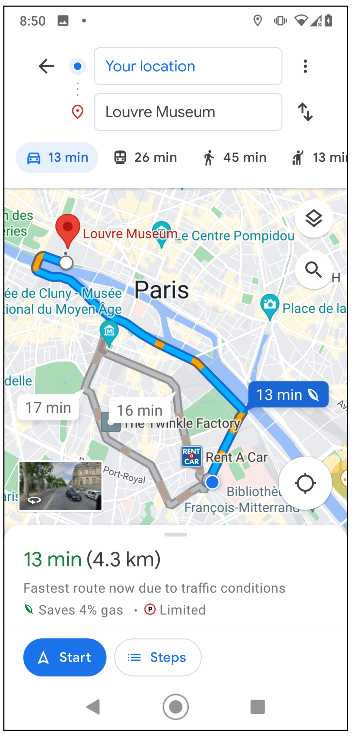
67 — 27 min
9:57 AM – 10:24 AM €2.50
Scheduled at 9:59 AM, 10:00 AM from Jenner - Jeanne d'Arc

10 › M 7 — 25 min
9:52 AM – 10:17 AM €2.10
Scheduled at 10:02 AM, 10:07 AM from Place d'Italie

M 5 › M 1 — 17 min
9:50 AM – 10:07 AM €2.10
Scheduled at 9:53 AM, 9:59 AM from Saint-Marcel

9 › 27 — 30 min
9:52 AM – 10:22 AM €2.50
Scheduled at 10:01 AM, 10:11 AM from Banquier

(B) Google Maps searching for public transport with fake location



13 min (4.3 km)
Fastest route now due to traffic conditions
Saves 4% gas · Limited
Start Steps

(C) Calculating a Google Maps route using a fictitious location

FIGURE 6.3: Testing the Google Maps services with a fictional location

6.2 Facebook

Social media has become an integral aspect of the daily lives of millennials in the digital age. Among the social media platforms extensively utilized by millennials is *Facebook*, boasting over 2 billion active users. This firmly establishes *Facebook* as a cornerstone of social media widely embraced by its users [103].

According to *Facebook's* LBS Terms, this application may collect and use the user's location data for purposes such as providing personalised content and information, enabling users to share their current and past locations with others, conducting measurement and analytics, offering various business services, promoting safety, conducting research and innovation for social benefit, and enhancing its products [104].

While *Facebook* collects location data and provides users with various features to share their location data, there is, however, the possibility of these data being exploited in a malicious manner. For instance, a digital stalker could utilise *Facebook's* profile viewing options to ascertain that their location has been tagged in a status, subsequently giving them the option to locate the individual or even go to their current location, which has been shared as a status [105].

Taking into consideration the previously mentioned, we chose to test the *Facebook* application with our solution, aiming to understand if it would be possible to provide fictitious location data, thereby altering its functionality regarding location-based application services.

Facebook has a feature called *memories* that allows users to review or share meaningful content from their past experiences on the platform. These *memories* can include the user's location data, provided that the user enables access to their location data.

To test the capability of our solution to provide *Facebook* with fake location data, the following actions were performed:

1. The *Facebook* application was launched without any restrictions on user location data access. We navigated to the *memories* option and selected the option to add location data to a *memory*. We observed that *Facebook* suggested locations around the user's real location.

2. Next, using the *XPrivacyLua* module, we blocked *Facebook's* access to our location data while providing fictitious location data previously generated using the *Mock-Location* application.

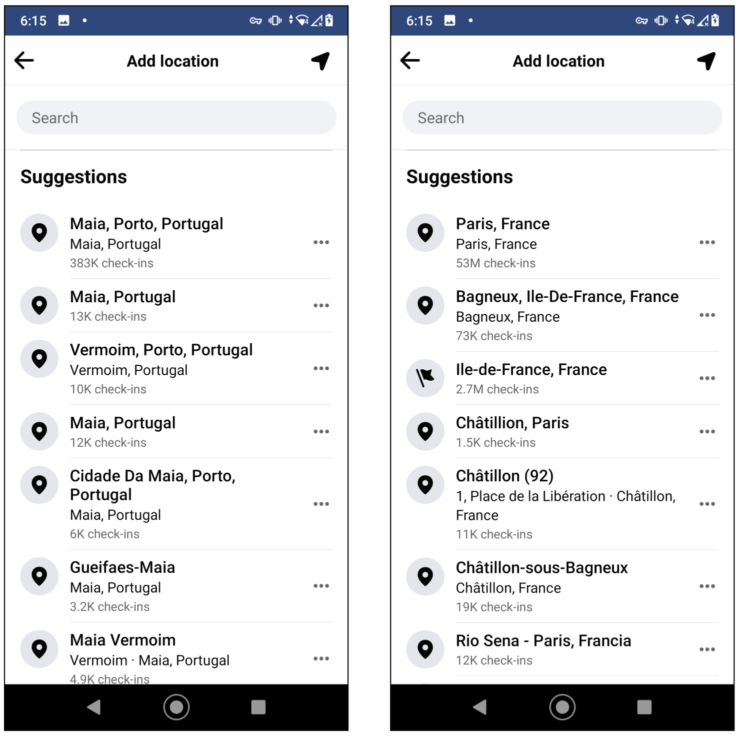
3. We relaunched the *Facebook* application and noticed that it still had access to the user's real location data.

Upon realising that our solution was not working for *Facebook*, we proceeded with debugging to identify the root cause of the issue. Analysis of the *logs* revealed that the script `location_createfromparcel` was unable to read the fictitious coordinates stored in the device memory.

The log analysis also indicated that the *Facebook* application uses multiple *threads* to access location data, leading us to consider that we might be facing a concurrency issue among these *threads* when accessing the memory location containing the fictitious coordinates.

To validate our hypothesis, we modified the `location_createfromparcel.lua` script by directly writing the values of the fictitious coordinates into the script code.

We then repeated the entire process described earlier and observed that the *Facebook* application now displayed the fictitious location data that we had *hardcoded* into the `location_createfromparcel.lua` script, as shown in Figure 6.4.


6:15
Add location
Search
Suggestions
Maia, Porto, Portugal
Maia, Portugal
383K check-ins
Maia, Portugal
13K check-ins
Vermoim, Porto, Portugal
Vermoim, Portugal
10K check-ins
Maia, Portugal
12K check-ins
Cidade Da Maia, Porto, Portugal
Maia, Portugal
6K check-ins
Gueifaes-Maia
Maia, Portugal
3.2K check-ins
Maia Vermoim
Vermoim · Maia, Portugal
4.9K check-ins

6:15
Add location
Search
Suggestions
Paris, France
Paris, France
53M check-ins
Bagneux, Ile-De-France, France
Bagneux, France
73K check-ins
Ile-de-France, France
2.7M check-ins
Châtillion, Paris
1.5K check-ins
Châtillon (92)
1, Place de la Libération · Châtillon, France
11K check-ins
Châtillon-sous-Bagneux
Châtillon, France
19K check-ins
Rio Sena - Paris, Francia
12K check-ins

(A) Facebook accessing real location

(B) Facebook accessing fake location data

FIGURE 6.4: Providing fake location data to Facebook using our solution

Testing our solution with the *Facebook* application revealed that our solution likely suffers from a concurrency issue among *threads* when reading the device memory where user-defined fictitious coordinates are stored. Despite this concurrency problem, which prevents us from using our solution with all its functionalities, we were still able to provide fake *static* locations to *Facebook* without compromising the application's function.

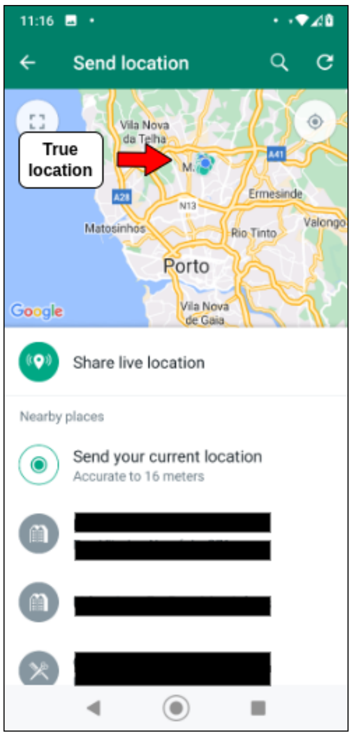
6.3 WhatsApp

WhatsApp is a cross-platform instant messaging application for smartphones. It allows users to send and receive location information, images, videos, audio, and text messages in *real-time* to individuals and groups of friends, all at no cost [106].

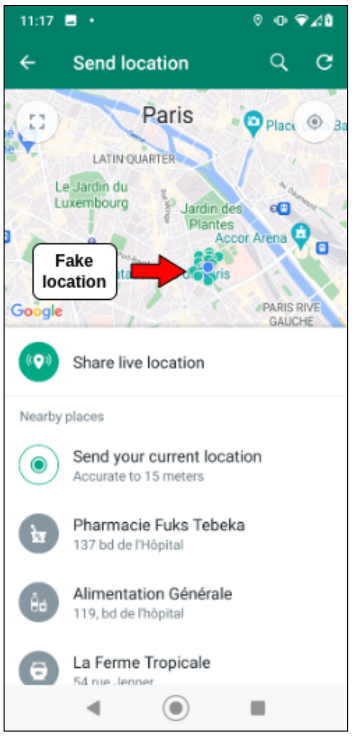
WhatsApp, with over 1.3 billion users across more than 180 countries, is widely regarded as one of the world's most popular mobile applications. This free online messaging service is owned by *Facebook*. Additionally, due to the increasing trend of remote work, *WhatsApp* has emerged as one of the primary communication tools in use today[107].

Considering the significance of *WhatsApp* in the realm of Android device communications, we chose to test our solution with this application.

Given that *WhatsApp* allows the *real-time* sharing of location with other users, our test, which yielded a positive outcome, aimed to verify whether the use of our solution enabled the sending of false location information to other users through the location-sharing feature provided by this application. Figure 6.5 presents the results of the tests conducted on *WhatsApp* using our solution which involved providing a fake *static* location.



(A) WhatsApp accessing real location



(B) WhatsApp accessing fake location data

FIGURE 6.5: Providing fake location data to WhatsApp using our solution

Our test was conducted as follows:

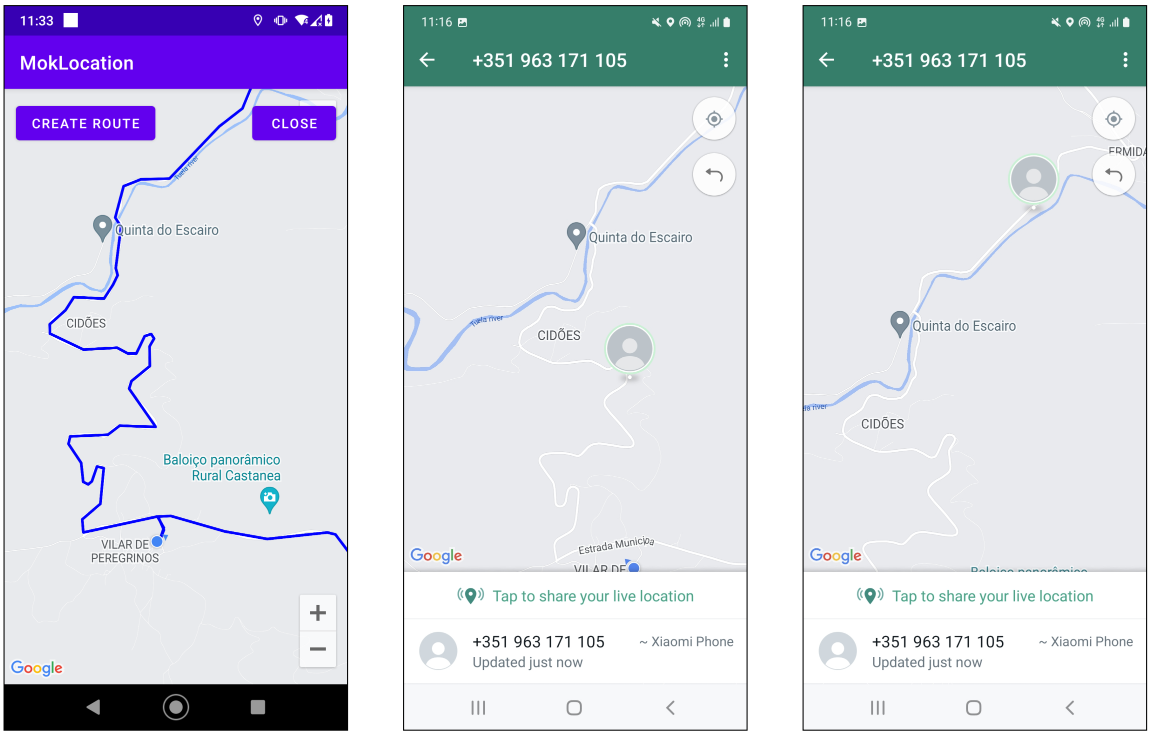
1. The *WhatsApp* application was launched without any controls in place regarding access to the actual device location. We observed that *WhatsApp* was accessing the user's real location (see Figure 6.7a).

2. Subsequently, through the *XPrivacyLua* module, access to the user's real location was blocked, and fictitious *static* location data created by the *MockLocation* application was provided (see Figure 6.2b).

3. Following the blocking of the user's real location, it was observed that when the *real-time* location-sharing feature was requested, *WhatsApp* proceeded to send fictitious location data (see Figure 6.7b).

In addition to allowing location sharing at a specific moment, *WhatsApp* has the functionality to share location permanently for a specified period determined by the user.

Considering the positive result obtained in the test with a *static* fake location, we proceeded to test our solution to simulate a car journey, sharing our fake location in *real-time* with another *WhatsApp* user. The results of the tests conducted with *real-time* location sharing in the *WhatsApp* application are shown in Figure 6.6.



(A) Creating a simulated route in the MockLocation

(B) Sharing location from the simulated route

(C) Sharing another location from the simulated route

FIGURE 6.6: Testing WhatsApp with simulated routes

The test with the simulated route had the following steps:

1. We simulated a car route using the MockLocation application (see Figure 6.6a).

2. Next, we launched the *WhatsApp* application and shared the fake location in *real-time* with another user. It was possible to observe that the location sent by *WhatsApp* to the chosen user for location sharing corresponded to points located on the simulated route (see Figure 6.6b and Figure 6.6c).

The tests conducted enabled us to observe the possibility of using our solution to provide fake *static* and *dynamic* location data to the *WhatsApp* application, while also allowing the sharing of this data with other users of the application.

6.4 Google Chrome

With a market share of 66.22% [108] on a global scale as of November 2022, *Google Chrome* is currently the most widely used mobile browser worldwide. To provide a personalised service, this application collects user location data [109], offering content and information specifically tailored to the user's current location.

To test our solution with the *Google Chrome* application, we followed the following procedure:

1. We launched the *Google Chrome* application without any restrictions regarding access to the user's true location.

2. Next, we blocked access to location data using the *XPrivacyLua* module and provided the application with fake location data that we had created previously using the *MockLocation* application (see Figure 6.2b). We observed that *Google Chrome* started determining the device's location solely based on IP address information, completely disregarding our fake location data. This test was conducted while connected to both a WiFi network and a mobile data connection.

Based on the results obtained, restricting access to the user's actual location data for this application can be achieved by combining our solution with a Virtual Private Network (VPN) [1] service that allows concealing the true IP address from which the device is accessing the internet.

[1] A VPN is a technology that provides secure and private internet connectivity by encrypting data traffic and routing it through servers located in different geographical regions. VPN are commonly used for safeguarding personal information, bypassing geo-restrictions, and ensuring a safer online experience.

6.5 Uber

In 2022, the *Uber* app was the second-most downloaded application worldwide, with 107 million downloads [102].

The access of this application to user location data is justified by the need to determine their location in order to provide a transportation service by matching the user's request with a nearby vehicle driver. However, *Uber's* access to user location data extends beyond providing the transportation service to the user.

This application implemented a controversial method that allowed the ride-hailing app to track the location of customers even when the application is running in the background. Before this tool was employed, *Uber* only collected user data when the app was actively open. Nevertheless, with this practice, *Uber* could gather location data for up to five minutes after the rider had closed the app, implying that *Uber* was attempting to ascertain the customer's final destination[110].

Keeping in mind that this type of application relies on user location access for its proper functioning, the fact that it accesses user data beyond the necessary period for service provision has made this application a candidate for testing with our solution. Thus, to test if we could provide fictitious location data to the *Uber* application, we carried out the following actions:

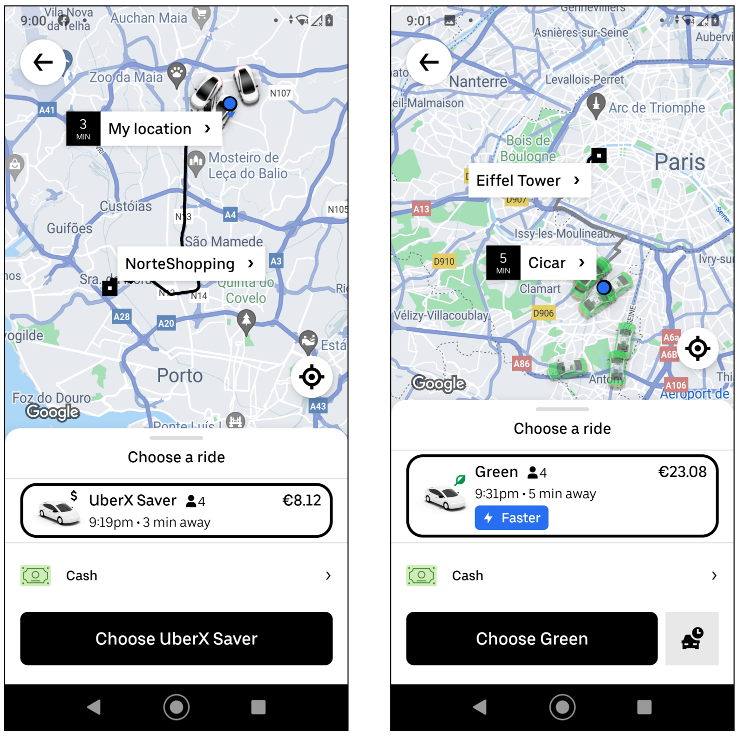
1. We launched the application without any restrictions on location data access and then selected a destination. We observed that the application calculated the trip using the user's real location as the starting point (see Figure 6.7a).

2. Next, we proceeded to block location data access to the Uber application using the *XPrivacyLua* module. Similar to what happened in our tests with the *Facebook* application, here too, the script `location_createfromparcel.lua` was unable to read the fictitious coordinates from the device memory.

Upon realising that we were facing a situation similar to what occurred during the *Facebook* application test, we modified the script `location_createfromparcel.lua`, rewriting the fictitious coordinates directly into the script's source code.

After making changes to the script `locate_createfromparcel.lua`, we re-ran the *Uber* application. When selecting a destination, the application calculated the route, using the *hardcoded* fictitious coordinates from the script `location_createfromparcel.lua` as the user's location (see Figure 6.7b).

In order to understand if the issue of concurrency, where the fictitious coordinates are stored, we proceeded to debug our solution. During this process, we found that the *Uber* application utilises a *multi-threaded* approach in accessing location data, resulting in a malfunction of our solution. The test results for the *Uber* application are displayed in Figure 6.7.



(A) Uber accessing real location

(B) Uber accessing fake location data

FIGURE 6.7: Providing fake location data to Uber using our solution

6.6 Reflection on the Results

Throughout this section, we describe the tests conducted on our solution to assess whether the initial objectives had been achieved. Considering that our solution was designed as a *proof of concept* and developed to provide both *static* and *dynamic* location data to the *Google Maps* application, our initial objectives have been met.

In addition to the *Google Maps* application, we were able to verify that our solution yielded positive results with the *WhatsApp* application. However, with the remaining tested applications, we obtained partially satisfactory results.

Race condition issues in accessing the memory location where fictitious coordinates are stored hindered the correct functioning of our solution in applications that use a *multi-threaded* approach to access user data.

A possible resolution to the *race condition* problem could be achieved by modifying the code of the *background service* executed by the *XPrivacyLua* module. Instead of storing the value of fictitious coordinates in a shared memory location to be read with each execution of the `location_createfromparcel.lua` script, we can utilise *global variables* accessible by the Lua script responsible for *hooking* the methods used by applications to access device location data.

Apart from the applications mentioned in this section, similar tests were conducted, with results comparable to those obtained for *Facebook* and *Uber*, on the following applications:

Strava

Tinder

AccuWeather

Waze

Considering that the tests on these applications yielded results similar to those obtained with the *Facebook* and *Uber* applications, we have chosen not to provide a detailed description of them in this section.

The possibility of developing a solution that allows for the provision of false location data, both statically and dynamically, to all applications installed on an Android device is strongly limited by the complexity of the Android operating system, which implements robust mechanisms such as *sandboxing* to restrict access to data manipulated by different applications.

Chapter 7

Conclusion

Despite the core of the operating system continuing to be released as part of the Android Open Source Project, the majority of applications that come pre-installed by Google and some manufacturers on Android devices are not open source. This situation is exacerbated by the increasing number of libraries and APIs that are only available on devices running pre-installed Google apps, making the overwhelming majority of third-party applications dependent on the Google ecosystem. Furthermore, it is not possible to uninstall these applications without obtaining administrative privileges (*root*), which are restricted by default for users. For these reasons, Android is described as a kind of open system where you can *"look but don't touch"* [111].

In addition to not allowing access to the system with administrator privileges, regarding user privacy protection, especially in controlling applications access to user location data, Android implements a permission system that proves insufficient in achieving the objectives for which it was designed, as described in Section 2.1.2.

The Android permission system is based on a *binary logic* where the user can only choose between two possible options - granting access or denying access to their location data. This model, while effective in blocking access to location data, limits the use of certain applications whose functionality depends on access to this data. In these applications, by revoking the permission to access location data, the user becomes unable to continue using the application.

The mechanisms implemented by Android for granting and revoking permissions to access location data by various applications that the user has installed on their device are often ignored by users who perceive this process as too cumbersome relative to their user experience [112] [113].

With privacy concerns of users gaining increasing attention in the media, as well as issues of illegitimate data collection and abuse of dominant position by Google, some open-source software enthusiasts have been developing solutions to safeguard user privacy. One of the most promising solutions currently in development is called the *microG Project* [111], which aims to provide an Android operating system without the need for Google's applications or its APIs.

Given the impossibility of undertaking a project as extensive as the *microG Project*, our solution focuses on creating mechanisms that, first and foremost, allow obtaining administrative privileges on the device and also control access to the user's location data by installed applications, with a particular emphasis on *Google Maps*, which is undoubtedly the most well-known and widely used navigation application globally.

Our solution aims to provide a mechanism that frees the user from the need to manage permissions to access location data through the mechanisms provided by the operating system, offering an alternative through which it is possible to provide fake location data to any application installed on the device without compromising its normal functioning and without the need to revoke any previously granted permissions.

7.1 Research Summary

Research conducted on the topic of user privacy in Android devices, with a special focus on location data, has shown that investigations related to this theme have been ongoing since the inception of Android, highlighting the significance of this issue in the context of mobile devices.

Within the extensive literature available on this subject, three significantly different approaches can be distinguished:

1. Solutions that monitor the data sent by applications outside the device have been studied. These solutions block, alter, or anonymize the data that leaves the device, preventing the inference of the user's true location.

2. The research conducted has revealed another type of approach focused on making app permissions more granular, replacing the permission system implemented by the Android operating system.

3. Finally, approaches that involve modifying the operating system itself to block access to location data and provide fake data to installed applications on the device.

Considering that users should have the ability to selectively determine when and which applications can access their actual location, and in cases where it is advantageous to provide fake data to an application for user safety, such as in cases of cyberstalking as mentioned in Section 6.2 regarding the *Facebook* application, we have chosen to create a solution that allows blocking access to the user's real location data and also allows the user to arbitrarily define a fictional location, which can be a *static* location or a simulated route.

7.2 Current Limitations

Given the ongoing evolution of the Android ecosystem, any solution requires constant maintenance and development. The *EdXposed* framework, which is used in our solution, is only compatible with devices running Android versions 8.0 to 11. In addition to this limitation, there has been no source code update for this framework from its developers since June 2021, which suggests that this project may be on the verge of being abandoned.

For the latest Android models (8.1 - 12, 12L DP1), the *LSPosed* [114] framework has emerged, offering the same functionalities as its predecessor, the *EdXposed* framework.

Although the *rooting* process is becoming increasingly simplified and, as a result, more accessible to a greater number of users, it remains a task that is beyond the reach of most users. Additionally, manufacturers do not provide warranty coverage for *rooted* devices in case of malfunctions.

The use of the device's shared memory for manipulating fictitious location data presents a dual limitation. On one hand, it generates *race condition* problems, restricting the functionality of our solution with applications that employ a *multi-threaded* approach to access location data. Moreover, it significantly elevates battery consumption due to its high processing demands, thereby limiting the temporal usage of our solution.

7.3 Future Work

During the completion of this dissertation, it was not possible to carry out some tasks and test certain ideas that arose due to time constraints in the development of this work.

In this section, we describe the tasks, improvements, and ideas that have been deferred to future work due to the lack of time.

While testing our solution with various applications, we observed instances of *race condition* issues in the script `location_createfromparcel.lua` accessing the device shared memory where fictitious coordinates are stored for supply to the applications. One possible solution to this problem would involve modifying the source code of the *background service* in the *XPrivacyLua* module and storing these values in class variables, accessible from the Lua script. This change would also help reduce battery consumption by requiring fewer computational resources.

Considering that the requirement of having a *rooted* device may deter a significant portion of users from adopting this solution, this limitation could potentially be overcome by using *VirtualXposed* [115]. Although it does not offer the same level of capability as *EdXposed*, it allows the use of *EdXposed* framework modules without the need to *root* the device.

Considering the existence of *LSPosed* for the latest Android versions, implementing our solution using this framework would allow us to encompass models equipped with the most recent Android versions.

In order to enhance the user experience of our solution, the creation of fictitious location data (routes and *static* points) performed through the *MockLocation* application should be integrated into the *XPrivacyLua* module, consolidating all the functionalities of our solution into a single application.

Our solution requires the user to be proactive in controlling access to their location data whenever they perceive that the knowledge of their true location by a particular application could threaten their privacy. The implementation of a mechanism that autonomously generates random fake location data whenever the user is in an area considered sensitive in terms of their privacy would free the user from the need to activate location data access controls every time they visit such places.

Finally, in order to provide a more realistic appearance to the fake location data, implementing fake accelerometer data on the device, correlated with the location data provided by our solution, would expand the usage spectrum of our solution, especially in fitness applications and those that rely on accelerometer data to infer the user's location.

7.4 Conclusions

During the course of our research, we addressed the main limitations of existing solutions in the context of controlling access to user location data by applications on Android devices.

The development of our solution was guided by the need to provide users with the ability to customise fictitious location data to be supplied to applications, allowing them to choose between *static* data or simulated routes.

Although it is just a *functional prototype*, our solution fulfills the objective of providing a *proof of concept* regarding the issue of application access to user location data.

While true location data is essential for effectively using the services that applications offer most of the time, there are situations in which users may benefit in terms of privacy and security by having the option to provide fictitious location data to certain applications. For example, this could help prevent situations such as cyberstalking or the abusive collection of data for profiling purposes, or to establish a system of penalties for citizens, as has been observed in countries with authoritarian political regimes.

Appendix A

Changes made to the XprivacyLua module code

In this appendix, we present the code of the XPrivacyLua module that has been modified and added during the course of this dissertation work.

A.1 Changes to the script `location_createfromparcel.lua`

```lua
-- This file is a modified version of the
-- location_createfromparcel.lua file from the XPrivacyLua
-- module.

function after(hook, param)
  local result = param:getResult()
  if result == nil then
  return false
end
  local old = result:toStrin
  -- start of added code
  local fakeCords = nil
  local file_path = nil
    "/sdcard/Android/mokLocation/fake_coordinates.txt"
  if io.open(file_path) then
```

```
      local file = io.open(file_path, "r")
       for line in io.lines(file_path) do
         fakeCords = line
         break
       end
      file:close()
  else
     print("File not found")
  end

  local fLatitude = nil
  local fLongitude =
  -- Extract lat and long values
  fLatitude, fLongitude =
    string.match(fakeCords, "^%s*(.-)%s*,%s*(.-)%s*$")
  local latitude = fLatitude
  local longitude = fLongit
  local type = param:getSetting('location.type')
  if type == 'set' then
    latitude = param:getSetting('location.latitude')
    longitude = param:getSetting('location.longitude')
    if latitude == nil or longitude == nil then
      latitude = fLatitude
      longitude = fLongitude
      end
  -- end of added code
  elseif type == 'coarse' then
    local accuracy = param:getSetting('location.accuracy')
    if accuracy ~= nil then
      local clatitude = param:getValue('latitude', hook)
      local clongitude = param:getValue('longitude', hook)
      if clatitude == nil or clongitude == nil then
        clatitude, clongitude =
          randomoffset(result:getLatitude(),
            result:getLongitude(), accuracy)
        param:putValue('latitude', clatitude, hook)
        param:putValue('longitude', clongitude, hook)
      end
      latitude = clatitude
      longitude = clongitude
    end
  end

  if result:hasAccuracy() then
    local accuracy = result:getAccuracy()
    if accuracy > 0 then
      latitude, longitude =
        randomoffset(latitude, longitude, accuracy)
      end
  end

  result:setLatitude(latitude)
  result:setLongitude(longitude)
    return true, old, result:toString()
end

function randomoffset(latitude, longitude, radius)
  local r = radius / 111000; -- degrees
  local w = r * math.sqrt(math.random())
  local t = 2 * math.pi * math.random()
  local lonoff = w * math.cos(t)
  local latoff = w * math.sin(t)
  lonoff = lonoff / math.cos(math.rad(latitude))

  return latitude + latoff, longitude + lonoff
end
```

LISTING A.1: Lua code for the script location_createfromparcel.lua

A.2 Java class for the background service

```java
package eu.faircode.xlua;

import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.Service;
import android.content.Intent;
import android.os.Build;
import android.os.IBinder;
import android.util.Log;
import androidx.annotation.Nullable;
import androidx.core.app.NotificationCompat;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;

public class BackGroundService extends Service {
  private static final int NOTIFICATION_ID = 1;
  private static final String CHANNEL_ID =
    "BackGroundServiceChannel";
  private String codeWord = "";
  private String last_cw = "";

  @Override
  public void onCreate() {
    super.onCreate();
  }

  @Override
  public int onStartCommand(Intent intent, int flags,
    int startId) {
    createNotificationChannel();

    NotificationCompat.Builder builder =
      new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_launcher_background)
        .setContentTitle("My Background Service")
        .setContentText("Running in the background");

    Notification notification = builder.build();
    startForeground(NOTIFICATION_ID, notification);

    // Schedule a task to read the code word
    // from the file every 5 seconds
    Timer setCodeWordTimer = new Timer();
    setCodeWordTimer.scheduleAtFixedRate(new TimerTask() {
      @Override
      public void run() {
        try {
          File file =
            new File("/sdcard/Android/mokLocation/
              codeWord.txt");
          BufferedReader reader =
            new BufferedReader(new FileReader(file));
          String line = reader.readLine();
          reader.close();
          codeWord = line;
          Log.d("ccd", "Set code word >>> " + codeWord);
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }, 0, 5000);

    TimerTask task = new TimerTask() {
      private Thread currentThread = null;
      @Override
      public void run() {
        try {
          File file =
            new File("/sdcard/Android/mokLocation/
              codeWord.txt");
          BufferedReader reader =
            new BufferedReader(new FileReader(file));
          String line = reader.readLine();
          reader.close();
          Log.d("ccd", "Check CodeWord >>> " + codeWord);
          Log.d("ccd", "Check Last CodeWord >>> " + last_cw);
          if (!line.equals(last_cw)) {
            last_cw = line;
            switch (line) {
              case "USER_CURRENT_LOCATION":
                // Stop the previous thread if it exists
                if (currentThread != null) {
                  currentThread.interrupt();
                }
                // Start a new thread to
                //handle user current location
                currentThread = new UserCurrentLocationThread();
                currentThread.start();
                break;
              case "DRIVING":
                // Stop the previous thread if it exists
                if (currentThread != null) {
                  currentThread.interrupt();
                }
                // Start a new thread to handle driving
                currentThread = new DrivingThread();
                currentThread.start();
                break;
              case "WALKING":
                // Stop the previous thread if it exists
                if (currentThread != null) {
                  currentThread.interrupt();
                }
                // Start a new thread to handle walking
                currentThread = new WalkingThread();
                currentThread.start();
                break;
              case "STATIC_POINT":
                // Stop the previous thread if it exists
                if (currentThread != null) {
                  currentThread.interrupt();
                }
                // Start a new thread to handle static point
                currentThread = new StaticPointThread();
                currentThread.start();
                break;
              default:
                // Do nothing if the code word
                //is not recognized
                break;
            }
          }
```

```
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    };

    Timer timer = new Timer();
    timer.schedule(task, 0, 5000);
    return START_STICKY;
  }

  @Override
  public void onDestroy() {
    super.onDestroy();
    stopForeground(true);
  }

  @Override
  public void onTaskRemoved(Intent rootIntent) {
    super.onTaskRemoved(rootIntent);
    stopSelf();
  }

  @Nullable
  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }

  private void createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
      NotificationChannel channel = new NotificationChannel(
              CHANNEL_ID,
              "My Background Service Channel",
              NotificationManager.IMPORTANCE_DEFAULT
      );
      NotificationManager manager =
              getSystemService(NotificationManager.class);
      manager.createNotificationChannel(channel);
    }
  }
}
```

LISTING A.2: Java class code for the background service

A.3 Java class for the threads managing static fake location data

```
package eu.faircode.xlua;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class StaticPointThread extends Thread {
  private boolean isInterrupted = false;

  @Override
  public void run() {
    try {
      while (!isInterrupted) {
        // Read the location from the file
        File locationFile =
          new File("/sdcard/Android/mokLocation/
            location.txt");
        BufferedReader locationReader =
          new BufferedReader(new FileReader(locationFile));
        String location = locationReader.readLine();
        locationReader.close();
        // Write the location to the fake coordinates file
        File fakeCoordinatesFile =
          new File("/sdcard/Android/mokLocation/
            fake_coordinates.txt");
        BufferedWriter fakeCoordinatesWriter =
          new BufferedWriter(
            new FileWriter(fakeCoordinatesFile));
        fakeCoordinatesWriter.write(location);
        fakeCoordinatesWriter.newLine();
        fakeCoordinatesWriter.close();
        // Wait for 5 seconds before
        //reading and writing again
        Thread.sleep(5000);
      }
    } catch (InterruptedException e) {
      // Thread was interrupted, exit gracefully
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  @Override
  public void interrupt() {
    isInterrupted = true;
    super.interrupt();
  }
}
```

LISTING A.3: Java code for the thread that manages location data for the user-chosen static point

```
package eu.faircode.xlua;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class UserCurrentLocationThread extends Thread {
  private boolean isInterrupted = false;
  @Override
  public void run() {
    try {
      while (!isInterrupted) {
        // Read the location from the file
        File locationFile =
          new File("/sdcard/Android/mokLocation/
            location.txt");
        BufferedReader locationReader =
          new BufferedReader(new FileReader(locationFile));
        String location = locationReader.readLine();
        locationReader.close();
        // Write the location to the fake coordinates file
        File fakeCoordinatesFile =
          new File("/sdcard/Android/mokLocation/
            fake_coordinates.txt");
        BufferedWriter fakeCoordinatesWriter =
          new BufferedWriter(
            new FileWriter(fakeCoordinatesFile));
        fakeCoordinatesWriter.write(location);
        fakeCoordinatesWriter.newLine();
        fakeCoordinatesWriter.close();
```

```
          // Wait for 5 seconds before
          //reading and writing again
          Thread.sleep(5000);
        }
      } catch (InterruptedException e) {
        // Thread was interrupted, exit gracefully
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
      @Override
    public void interrupt() {
      isInterrupted = true;
      super.interrupt();
    }
}
```

LISTING A.4: Java code for the thread that manages location data for the user-current location

A.4 Java class for the threads managing dynamic fake location data

```
package eu.faircode.xlua;

import android.os.Environment;
import android.util.Log;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;


public class DrivingThread extends Thread {
  private  boolean isInterrupted = false;


  @Override
  public void run() {
    simulateDrivingRoute();
  }

  @Override
  public void interrupt() {
    isInterrupted = true;
    super.interrupt();
  }

  public void setIsInterrupted() {
    isInterrupted = false;
  }

  public void simulateDrivingRoute() {
    String waypointsFilePath =
      "/sdcard/Android/mokLocation/location_driving_route.txt";
    int segmentsPerWaypoint = 5;
    int timeInterval = 200;
    try {
      interpolatePathToFile(waypointsFilePath,
      segmentsPerWaypoint, timeInterval);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  // Interpolates positions between waypoints to
  //create a smooth path and writes the
  //simulated coordinates to a file

  public  void interpolatePathToFile(String waypointsFilePath,
    int segmentsPerWaypoint, int timeInterval)
      throws IOException {

    List<String> waypoints =
      readWaypointsFromFile(waypointsFilePath);
    List<String> path =
      interpolatePath(waypoints, segmentsPerWaypoint);
    writeCoordinatesToFile(path, timeInterval);
  }

  // Reads waypoints from a file
  private static List<String>
    readWaypointsFromFile(String filePath)
      throws IOException {
    List<String> waypoints = new ArrayList<>();
    try (BufferedReader reader =
      new BufferedReader(new FileReader(filePath))) {
      String line;
      while ((line = reader.readLine()) != null) {
        waypoints.add(line);
      }
    }
    return waypoints;
  }

  private  void writeCoordinatesToFile(List<String> path,
  int timeInterval) throws IOException {
    for (int i = 0; i < path.size(); i++) {
      Log.d("ccd", path.get(i));
      Log.d("ccd",
        "isInterrupted value driving route >>> " +
          isInterrupted);
      if(isInterrupted == true){
        return;
      }
      writeLocationCodeToFile(path.get(i));
      if (i < path.size() - 1) {
        try {
          Thread.sleep(timeInterval);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
  }

  private static void writeLocationCodeToFile(
    String locationCodeWord){

    String data = locationCodeWord + "\n\r";
    // Create the file to store the user static location
    File sdCard = Environment.getExternalStorageDirectory();
    // Create a new file with the specified name and path
    String storageDir =
      sdCard.getAbsolutePath() + "/Android/mokLocation";
    File file = new File(storageDir, "fake_coordinates.txt");

    if (file.exists()){
    }
    if (!file.exists()) {
      try {
        file.createNewFile();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
    try {
      FileWriter writer = new FileWriter(file);
      writer.write(data);
      writer.flush();
      writer.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

    // Interpolates positions between
    //waypoints to create a smooth path
  public static List<String>
    interpolatePath(List<String> waypoints,
    int segmentsPerWaypoint) {

    List<String> path = new ArrayList<String>();
    for (int i = 0; i < waypoints.size() - 1; i++) {
      String startPoint = waypoints.get(i);
      String endPoint = waypoints.get(i + 1);
      double[] startCoords = parseCoords(startPoint);
      double[] endCoords = parseCoords(endPoint);
      double distance =
        haversine(startCoords[0], startCoords[1],
          endCoords[0], endCoords[1]);
      double segmentDistance = distance / segmentsPerWaypoint;
      for (int j = 0; j < segmentsPerWaypoint; j++) {
        double t = (double) j / (double) segmentsPerWaypoint;
        double[] coords = interpolate(startCoords, endCoords, t);
        path.add(formatCoords(coords[0], coords[1]));
      }
    }
    // Add last waypoint
    path.add(waypoints.get(waypoints.size() - 1));
    return path;
  }

  // Parses coordinates from a string
  //in the format "latitude, longitude"
  private static double[] parseCoords(String coordsString) {

    String[] parts = coordsString.split(",");
    double[] coords = new double[2];
    coords[0] = Double.parseDouble(parts[0].trim());
    coords[1] = Double.parseDouble(parts[1].trim());
    return coords;
  }

  // Formats coordinates as a string
  //in the format "latitude, longitude"
  private static String formatCoords(
    double latitude, double longitude){
    return String.format("%.6f, %.6f", latitude, longitude);
  }
```

```java
  // Interpolates between two points
  //using linear interpolation
  private static double[] interpolate(double[] startCoords,
    double[] endCoords, double t) {

    double latitude = startCoords[0] +
      (endCoords[0] - startCoords[0]) * t;
    double longitude = startCoords[1] +
      (endCoords[1] - startCoords[1]) * t;
    return new double[] { latitude, longitude };
  }

  // Calculates the great-circle distance
  //between two points using the Haversine formula
  private static double haversine(double lat1,
    double lon1, double lat2, double lon2) {

    final int R = 6371; // Earth's radius in km
    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    double a = Math.sin(latDistance / 2) *
      Math.sin(latDistance / 2)
            + Math.cos(Math.toRadians(lat1)) *
              Math.cos(Math.toRadians(lat2))
            * Math.sin(lonDistance / 2) *
              Math.sin(lonDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c;
    return distance;
  }
}
```

LISTING A.5: Java code for the thread that manages simulated driving route coordinates

```java
package eu.faircode.xlua;

import android.os.Environment;
import android.util.Log;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;


public class DrivingThread extends Thread {
  private  boolean isInterrupted = false;


  @Override
  public void run() {
    simulateDrivingRoute();
  }

  @Override
  public void interrupt() {
    isInterrupted = true;
    super.interrupt();
  }

  public void setIsInterrupted() {
    isInterrupted = false;
  }

  public void simulateDrivingRoute() {
    String waypointsFilePath =
      "/sdcard/Android/mokLocation/location_driving_route.txt";
    int segmentsPerWaypoint = 20;
    int timeInterval = 2000;
    try {
      interpolatePathToFile(waypointsFilePath,
      segmentsPerWaypoint, timeInterval);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  // Interpolates positions between waypoints to
  //create a smooth path and writes the
  //simulated coordinates to a file

  public  void interpolatePathToFile(String waypointsFilePath,
    int segmentsPerWaypoint, int timeInterval)
      throws IOException {

    List<String> waypoints =
      readWaypointsFromFile(waypointsFilePath);
    List<String> path =
      interpolatePath(waypoints, segmentsPerWaypoint);
    writeCoordinatesToFile(path, timeInterval);
  }

  // Reads waypoints from a file
  private static List<String>
```

```
    readWaypointsFromFile(String filePath)
      throws IOException {
    List<String> waypoints = new ArrayList<>();
    try (BufferedReader reader =
      new BufferedReader(new FileReader(filePath))) {
      String line;
      while ((line = reader.readLine()) != null) {
        waypoints.add(line);
      }
    }
    return waypoints;
  }

  private  void writeCoordinatesToFile(List<String> path,
  int timeInterval) throws IOException {
    for (int i = 0; i < path.size(); i++) {
      Log.d("ccd", path.get(i));
      Log.d("ccd",
        "isInterrupted value driving route >>> " +
          isInterrupted);
      if(isInterrupted == true){
        return;
      }
      writeLocationCodeToFile(path.get(i));
      if (i < path.size() - 1) {
        try {
          Thread.sleep(timeInterval);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
  }
```

```
private static void writeLocationCodeToFile(
  String locationCodeWord){

  String data = locationCodeWord + "\n\r";
  // Create the file to store the user static location
  File sdCard = Environment.getExternalStorageDirectory();
  // Create a new file with the specified name and path
  String storageDir =
    sdCard.getAbsolutePath() + "/Android/mokLocation";
  File file = new File(storageDir, "fake_coordinates.txt");

  if (file.exists()){
  }
  if (!file.exists()) {
    try {
      file.createNewFile();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  try {
    FileWriter writer = new FileWriter(file);
    writer.write(data);
    writer.flush();
    writer.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

  // Interpolates positions between
  //waypoints to create a smooth path
```

```
public static List<String>
  interpolatePath(List<String> waypoints,
  int segmentsPerWaypoint) {

  List<String> path = new ArrayList<String>();
  for (int i = 0; i < waypoints.size() - 1; i++) {
    String startPoint = waypoints.get(i);
    String endPoint = waypoints.get(i + 1);
    double[] startCoords = parseCoords(startPoint);
    double[] endCoords = parseCoords(endPoint);
    double distance =
      haversine(startCoords[0], startCoords[1],
        endCoords[0], endCoords[1]);
    double segmentDistance = distance / segmentsPerWaypoint;
    for (int j = 0; j < segmentsPerWaypoint; j++) {
      double t = (double) j / (double) segmentsPerWaypoint;
      double[] coords = interpolate(startCoords,
        endCoords, t);
      path.add(formatCoords(coords[0], coords[1]));
    }
  }
  // Add last waypoint
  path.add(waypoints.get(waypoints.size() - 1));
  return path;
}

// Parses coordinates from a
//string in the format "latitude, longitude"
private static double[] parseCoords(String coordsString) {

  String[] parts = coordsString.split(",");
  double[] coords = new double[2];
  coords[0] = Double.parseDouble(parts[0].trim());
```

```
    coords[1] = Double.parseDouble(parts[1].trim());
    return coords;
  }

  // Formats coordinates as a string
  //in the format "latitude, longitude"
  private static String formatCoords(
    double latitude, double longitude){
    return String.format("%.6f, %.6f", latitude, longitude);
  }

  // Interpolates between two
  //points using linear interpolation
  private static double[] interpolate(double[] startCoords,
    double[] endCoords, double t) {

    double latitude = startCoords[0] +
      (endCoords[0] - startCoords[0]) * t;
    double longitude = startCoords[1] +
      (endCoords[1] - startCoords[1]) * t;
    return new double[] { latitude, longitude };
  }

  // Calculates the great-circle distance
  //between two points using the Haversine formula
  private static double haversine(double lat1,
    double lon1, double lat2, double lon2) {

    final int R = 6371; // Earth's radius in km
    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    double a = Math.sin(latDistance / 2) *
      Math.sin(latDistance / 2)
```

```
                + Math.cos(Math.toRadians(lat1)) *
                  Math.cos(Math.toRadians(lat2))
                * Math.sin(lonDistance / 2) *
                  Math.sin(lonDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c;
    return distance;
  }
}
```

LISTING A.6: Java code for the thread that manages simulated walking route coordinates

Bibliography

[1] I. The Radicati Group, "Mobile Statistics Report, 2021-2025," The Radicati Group, Inc. - A Technology Market Research Firm, Tech. Rep., 2021. [Online]. Available: https://www.radicati.com/wp/wp-content/uploads/2021/Mobile_Statistics_Report,_2021-2025_Executive_Summary.pdf [Cited on page 1.]

[2] Y.-K. Lee, C.-T. Chang, Y. Lin, and Z.-H. Cheng, "The dark side of smartphone usage: Psychological traits, compulsive behavior and technostress," *Computers in Human Behavior*, vol. 31, pp. 373–383, Feb. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S074756321300397X [Cited on page 1.]

[3] D. Rozgonjuk and J. Elhai, "Problematic smartphone usage, emotion regulation, and social and non-social smartphone use," in *Proceedings of the Technology, Mind, and Society*. Washington DC USA: ACM, Apr. 2018, pp. 1–1. [Online]. Available: https://dl.acm.org/doi/10.1145/3183654.3183664 [Cited on page 1.]

[4] H. Yun, D. Han, D.-R. Bundang-Gu, S.-S. Gyeonggi-do, and C. C. Lee, "UNDERSTANDING THE USE OF LOCATION-BASED SERVICE APPLICATIONS:," vol. 14, no. 3, 2013. [Cited on page 1.]

[5] F. Zhao, L. Gao, Y. Zhang, Z. Wang, B. Wang, and S. Guo, "You Are Where You App: An Assessment on Location Privacy of Social Applications," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2018, pp. 236–247, iSSN: 2332-6549. [Cited on page 2.]

[6] S. Y. Esayas, "The idea of 'emergent properties' in data privacy: towards a holistic approach," *International Journal of Law and Information Technology*, vol. 25, no. 2, pp. 139–178, 2017. [Cited on page 2.]

[7] J. Thatcher, "Big Data, Big Questions| Living on Fumes: Digital Footprints, Data Fumes, and the Limitations of Spatial Big Data," *International Journal of*

Communication, vol. 8, no. 0, p. 19, Jun. 2014, number: 0. [Online]. Available: https://ijoc.org/index.php/ijoc/article/view/2174 [Cited on page 2.]

[8] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: automated security validation of mobile apps at app markets," in *Proceedings of the second international workshop on Mobile cloud computing and services*, 2011, pp. 21–26. [Cited on page 2.]

[9] M. Madden, "Public Perceptions of Privacy and Security in the Post-Snowden Era," Nov. 2014. [Online]. Available: https://www.pewresearch.org/internet/2014/11/12/public-privacy-perceptions/ [Cited on page 2.]

[10] A. Sunyaev, T. Dehling, P. L. Taylor, and K. D. Mandl, "Availability and quality of mobile health app privacy policies," *Journal of the American Medical Informatics Association*, vol. 22, no. e1, pp. e28–e33, 2015. [Cited on page 2.]

[11] A. G. Ray, Sri, "Grindr Is Sharing The HIV Status Of Its Users With Other Companies," Apr. 2018, section: Science. [Online]. Available: https://www.buzzfeednews.com/article/azeenghorayshi/grindr-hiv-status-privacy [Cited on page 2.]

[12] A. M. McDonald and L. F. Cranor, "The cost of reading privacy policies," *Isjlp*, vol. 4, p. 543, 2008. [Cited on page 3.]

[13] P. B. Brandtzæg, M. Lüders, and J. H. Skjetne, ""Too many Facebook 'friends'? Content sharing and sociability versus the need for privacy in social network sites": Corrigenda," *International Journal of Human-Computer Interaction*, vol. 27, no. 1, pp. 106–106, 2011, place: United Kingdom Publisher: Taylor & Francis. [Cited on page 3.]

[14] A. Khatoon and P. Corcoran, "Android permission system and user privacy — A review of concept and approaches," in *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. Berlin: IEEE, Sep. 2017, pp. 153–158. [Online]. Available: http://ieeexplore.ieee.org/document/8210616/ [Cited on pages 3, 10, and 14.]

[15] H. Anwar, D. Pfahl, and S. N. Srirama, "Evaluating the impact of code smell refactoring on the energy consumption of android applications," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 82–86. [Cited on page 3.]

[16] "Google Play Store: number of apps 2023." [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ [Cited on page 3.]

[17] D. Hayes, F. Cappa, and N. A. Le-Khac, "An effective approach to mobile device management: Security and privacy issues associated with mobile applications," *Digital Business*, vol. 1, no. 1, p. 100001, Sep. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2666954420300016 [Cited on pages 3 and 14.]

[18] D. Barrera and P. Van Oorschot, "Secure software installation on smartphones," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 42–48, 2010. [Cited on page 3.]

[19] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011, pp. 239–252. [Cited on page 3.]

[20] K. Zickuhr, *Three-quarters of smartphone owners use location-based services*. Pew Internet & American Life Project, 2012. [Cited on page 3.]

[21] J. L. Boyles, A. Smith, and M. Madden, "Privacy and data management on mobile devices," *Pew Internet & American Life Project*, vol. 4, pp. 1–19, 2012. [Cited on page 3.]

[22] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the eighth symposium on usable privacy and security*, 2012, pp. 1–14. [Cited on page 3.]

[23] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A conundrum of permissions: installing applications on an android smartphone," in *Financial Cryptography and Data Security: FC 2012 Workshops, USEC and WECSR 2012, Kralendijk, Bonaire, March 2, 2012, Revised Selected Papers 16*. Springer, 2012, pp. 68–79. [Cited on page 3.]

[24] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2013, pp. 3393–3402. [Cited on page 3.]

[25] Y.-A. De Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the Crowd: The privacy bounds of human mobility," *Scientific Reports*, vol. 3, no. 1, p. 1376, Mar. 2013. [Online]. Available: https://www.nature.com/articles/srep01376 [Cited on page 5.]

[26] J. Pang and Y. Zhang, "Deepcity: A feature learning framework for mining location check-ins," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 11, no. 1, 2017, pp. 652–655. [Cited on page 5.]

[27] P. Golle and K. Partridge, "On the Anonymity of Home/Work Location Pairs," in *Pervasive Computing*, H. Tokuda, M. Beigl, A. Friday, A. J. B. Brush, and Y. Tobe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5538, pp. 390–397, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-01516-8_26 [Cited on pages 5 and 38.]

[28] S. Scellato, A. Noulas, and C. Mascolo, "Exploiting place features in link prediction on location-based social networks," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1046–1054. [Cited on page 5.]

[29] M. Backes, M. Humbert, J. Pang, and Y. Zhang, "walk2friends: Inferring social links from mobility profiles," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1943–1957. [Cited on page 5.]

[30] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao, "Whispers in the dark: analysis of an anonymous social network," in *Proceedings of the 2014 conference on internet measurement conference*, 2014, pp. 137–150. [Cited on page 5.]

[31] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones," *Communications of the ACM*, vol. 57, no. 3, pp. 99–106, Mar. 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2494522 [Cited on pages 5, 23, and 24.]

[32] H. Jiang, J. Li, P. Zhao, F. Zeng, Z. Xiao, and A. Iyengar, "Location Privacy-preserving Mechanisms in Location-based Services: A Comprehensive Survey," *ACM Computing Surveys*, vol. 54, no. 1, pp. 1–36, Jan. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3423165 [Cited on page 5.]

[33] J. Leyden, "Dark net LinkedIn sale looks like the real deal." [Online]. Available: https://www.theregister.com/2016/05/18/linkedin/ [Cited on page 5.]

[34] S. Perez, "Recently confirmed Myspace hack could be the largest yet," May 2016. [Online]. Available: https://techcrunch.com/2016/05/31/recently-confirmed-myspace-hack-could-be-the-largest-yet/ [Cited on page 5.]

[35] K. Drakonakis, P. Ilia, S. Ioannidis, and J. Polakis, "Please forget where i was last summer: The privacy risks of public location (meta) data," *arXiv preprint arXiv:1901.00897*, 2019. [Cited on page 5.]

[36] N. Eddy, "Location-Based Applications Popular, Despite Privacy Concerns: ISACA - Mobile and Wireless - News & Reviews - eWeek.com," Apr. 2012. [Online]. Available: https://www.eweek.com/mobile/location-based-applications-popular-despite-privacy-concerns-isaca/ [Cited on page 5.]

[37] B. Soewito and A. Suwandaru, "Android sensitive data leakage prevention with rooting detection using Java function hooking," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 5, pp. 1950–1957, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157820304201 [Cited on page 8.]

[38] Y. Shao, X. Luo, and C. Qian, "Rootguard: Protecting rooted android phones," *Computer*, vol. 47, no. 6, pp. 32–40, 2014. [Cited on page 8.]

[39] S.-T. Sun, A. Cuadros, and K. Beznosov, "Android rooting: Methods, detection, and evasion," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015, pp. 3–14. [Cited on page 8.]

[40] S. Garg and N. Baliyan, "Comparative analysis of Android and iOS from security viewpoint," *Computer Science Review*, vol. 40, p. 100372, May 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000125 [Cited on page 11.]

[41] "LineageOS Statistics." [Online]. Available: https://stats.lineageos.org/ [Cited on page 12.]

[42] "Titanium Backup (root needed) - Apps on Google Play." [Online]. Available: https://play.google.com/store/apps/details?id=com.keramidas.TitaniumBackup&hl=en_US [Cited on page 12.]

[43] "Explorer - Apps on Google Play." [Online]. Available: https://play.google.com/store/apps/details?id=com.speedsoftware.explorer&hl=en_US [Cited on pages 12 and 15.]

[44] S. Egelman, L. F. Cranor, and J. Hong, "You've been warned: an empirical study of the effectiveness of web browser phishing warnings," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2008, pp. 1065–1074. [Cited on page 13.]

[45] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of ssl warning effectiveness." in *USENIX security symposium*. Montreal, Canada, 2009, pp. 399–416. [Cited on page 13.]

[46] Google, "Google Report - Android Security 2014 Year in Review," Tech. Rep. [Online]. Available: https://source.android.com/static/docs/security/overview/reports/Google_Android_Security_2014_Report_Final.pdf [Cited on page 13.]

[47] S.-T. Sun, A. Cuadros, and K. Beznosov, "Android Rooting: Methods, Detection, and Evasion," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. Denver Colorado USA: ACM, Oct. 2015, pp. 3–14. [Online]. Available: https://dl.acm.org/doi/10.1145/2808117.2808126 [Cited on page 13.]

[48] "80% China's Mobile Users Rooted Smartphones in 2014," Apr. 2015. [Online]. Available: https://www.chinainternetwatch.com/12926/80-china-smartphone-users-rooted/ [Cited on page 13.]

[49] C. Stach, "How to Assure Privacy on Android Phones and Devices?" in *2013 IEEE 14th International Conference on Mobile Data Management*, vol. 1, Jun. 2013, pp. 350–352, iSSN: 2375-0324. [Cited on page 14.]

[50] J. Kim, Y. Yoon, K. Yi, and J. Shin, "SCANDAL: Static Analyzer for Detecting Privacy Leaks in Android Applications." [Cited on page 14.]

[51] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security." [Cited on page 14.]

[52] "App permissions best practices." [Online]. Available: https://developer.android.com/training/permissions/usage-notes [Cited on page 14.]

[53] C. Stach and B. Mitschang, "Design and Implementation of the Privacy Management Platform," in *2014 IEEE 15th International Conference on Mobile Data Management*, vol. 1, Jul. 2014, pp. 69–72, iSSN: 2375-0324. [Cited on page 14.]

[54] R. L. Finn, D. Wright, and M. Friedewald, "Seven Types of Privacy," in *European Data Protection: Coming of Age*, S. Gutwirth, R. Leenes, P. De Hert, and Y. Poullet, Eds. Dordrecht: Springer Netherlands, 2013, pp. 3–32. [Online]. Available: https://link.springer.com/10.1007/978-94-007-5170-5_1 [Cited on page 14.]

[55] K. Pelgrims, *Gradle for Android*. Packt Publishing Ltd, 2015. [Cited on page 14.]

[56] Oracle, "Using Java Reflection." [Online]. Available: https://www.oracle.com/technical-resources/articles/java/javareflection.html [Cited on page 15.]

[57] GeeksforGeeks, "Reflection in Java," Mar. 2016, section: Java. [Online]. Available: https://www.geeksforgeeks.org/reflection-in-java/ [Cited on page 15.]

[58] M. T. Serrafero, "Xposed: Best of XDA," Jun. 2015, section: XDA Android. [Online]. Available: https://www.xda-developers.com/xposed-best-of-xda/ [Cited on page 16.]

[59] rovo89, "rovo89/Xposed," Jul. 2023, original-date: 2012-03-25T13:40:18Z. [Online]. Available: https://github.com/rovo89/Xposed [Cited on page 16.]

[60] XDA-Forums, "[OFFICIAL] EdXposed - The successor of Xposed [Oreo/Pie/Q/R, 2020/07/19]," Mar. 2020. [Online]. Available: https://forum.xda-developers.com/t/official-edxposed-the-successor-of-xposed-oreo-pie-q-r-2020-07-19.4070199/ [Cited on page 16.]

[61] "PAGalaxyLab/YAHFA," Aug. 2023, original-date: 2017-03-30T06:45:05Z. [Online]. Available: https://github.com/PAGalaxyLab/YAHFA [Cited on pages 16 and 49.]

[62] S. Hazarika, "What is EdXposed, and what can you do with it on your Android device?" May 2022, section: Mobile. [Online]. Available: https://www.xda-developers.com/edxposed/ [Cited on page 16.]

[63] XDA-Developers, "How to Install Magisk on your Android Phone," Aug. 2021, section: Mobile. [Online]. Available: https://www.xda-developers.com/how-to-install-magisk/ [Cited on page 17.]

[64] A. Srivastava, "Android Xposed Framework," Aug. 2020. [Online]. Available: https://abhiappmobiledeveloper.medium.com/android-xposed-framework-ee763e5dda8e [Cited on page 17.]

[65] XDA-Developers, "What is Riru, and what can you do with it on your Android device?" Apr. 2022, section: Mobile. [Online]. Available: https://www.xda-developers.com/riru/ [Cited on page 17.]

[66] R. Ierusalimschy, *Programming in Lua, Fourth Edition*, 2016. [Cited on page 18.]

[67] L. Jordan and P. Greyling, "Embedding Lua in Android Applications," in *Practical Android Projects*. Berkeley, CA: Apress, 2011, pp. 155–192. [Online]. Available: http://link.springer.com/10.1007/978-1-4302-3244-5_4 [Cited on page 18.]

[68] "Roseborough.com." [Online]. Available: http://www.luaj.org/luaj.html [Cited on page 19.]

[69] K. Fawaz and K. G. Shin, "Location Privacy Protection for Smartphone Users," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. Scottsdale Arizona USA: ACM, Nov. 2014, pp. 239–250. [Online]. Available: https://dl.acm.org/doi/10.1145/2660267.2660270 [Cited on pages 20, 22, and 23.]

[70] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. Android and Mr. Hide: fine-grained permissions in android applications," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. Raleigh North Carolina USA: ACM, Oct. 2012, pp. 3–14. [Online]. Available: https://dl.acm.org/doi/10.1145/2381934.2381938 [Cited on pages 21 and 22.]

[71] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "MockDroid: trading privacy for application functionality on smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*. Phoenix Arizona: ACM, Mar. 2011, pp. 49–54. [Online]. Available: https://dl.acm.org/doi/10.1145/2184489.2184500 [Cited on page 23.]

[72] K. Fawaz, H. Feng, and K. G. Shin, "Anatomization and Protection of Mobile Apps' Location Privacy Threats," 2015. [Cited on pages 24 and 25.]

[73] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," 2011. [Cited on pages 26 and 27.]

[74] S. Guha, M. Jain, and V. N. Padmanabhan, "Koi: A Location-Privacy Platform for Smartphone Apps," 2012. [Cited on pages 27 and 28.]

[75] J. Kang, D. Steiert, D. Lin, and Y. Fu, "MoveWithMe: Location Privacy Preservation for Smartphone Users," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 711–724, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8759922/ [Cited on pages 29 and 30.]

[76] S. Chitkara, N. Gothoskar, S. Harish, J. I. Hong, and Y. Agarwal, "Does this App Really Need My Location?: Context-Aware Privacy Management for Smartphones," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 1–22, Sep. 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3132029 [Cited on pages 30 and 31.]

[77] J. Estrela, "Android Security by Introspection," Ph.D. dissertation, 2019. [Cited on page 32.]

[78] J. Krumm, "Realistic driving trips for location privacy," in *Pervasive Computing: 7th International Conference, Pervasive 2009, Nara, Japan, May 11-14, 2009. Proceedings 7*. Springer, 2009, pp. 25–41. [Cited on pages 33 and 36.]

[79] B. Palanisamy and L. Liu, "MobiMix: Protecting location privacy with mix-zones over road networks," in *2011 IEEE 27th International Conference on Data Engineering*, Apr. 2011, pp. 494–505, iSSN: 2375-026X. [Cited on page 33.]

[80] J. Meyerowitz and R. Roy Choudhury, "Hiding stars with fireworks: location privacy through camouflage," in *Proceedings of the 15th annual international conference on Mobile computing and networking*, ser. MobiCom '09. New York, NY, USA: Association for Computing Machinery, Sep. 2009, pp. 345–356. [Online]. Available: https://dl.acm.org/doi/10.1145/1614320.1614358 [Cited on page 34.]

[81] T. Hara, A. Suzuki, M. Iwata, Y. Arase, and X. Xie, "Dummy-Based User Location Anonymization Under Real-World Constraints," *IEEE Access*, vol. 4, pp. 673–687, 2016, conference Name: IEEE Access. [Cited on page 34.]

[82] V. Bindschaedler and R. Shokri, "Synthesizing Plausible Privacy-Preserving Location Traces," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 546–563, iSSN: 2375-1207. [Cited on page 35.]

[83] O. Jan, A. J. Horowitz, and Z.-R. Peng, "Using global positioning system data to understand variations in path choice," *Transportation Research Record*, vol. 1725, no. 1, pp. 37–44, 2000. [Cited on page 38.]

[84] J. Krumm, "Inference attacks on location tracks," in *Pervasive Computing: 5th International Conference, PERVASIVE 2007, Toronto, Canada, May 13-16, 2007. Proceedings 5*. Springer, 2007, pp. 127–143. [Cited on page 38.]

[85] M. Bokhorst, "XPrivacyLua," Aug. 2023, original-date: 2018-01-05T11:54:06Z. [Online]. Available: https://github.com/M66B/XPrivacyLua [Cited on pages 41 and 49.]

[86] G. B. Meike and L. Schiefer, *Inside the android OS: building, customizing, managing, and operating android system services*, 1st ed. Hoboken: Pearson Education, Inc, 2021. [Cited on pages 42 and 43.]

[87] "Development tutorial." [Online]. Available: https://github.com/rovo89/XposedBridge/wiki/Development-tutorial [Cited on page 43.]

[88] "XposedBridge | Xposed Framework API." [Online]. Available: https://api.xposed.info/reference/de/robv/android/xposed/XposedBridge.html [Cited on page 43.]

[89] "Android Debug Bridge (adb) | Android Studio." [Online]. Available: https://developer.android.com/tools/adb [Cited on page 46.]

[90] "TeamWin - TWRP." [Online]. Available: https://twrp.me/ [Cited on page 47.]

[91] L. Nguyen-Vu, N.-T. Chau, S. Kang, and S. Jung, "Android Rooting: An Arms Race between Evasion and Detection," *Security and Communication Networks*, vol. 2017, pp. 1–13, 2017. [Online]. Available: https://www.hindawi.com/journals/scn/2017/4121765/ [Cited on page 47.]

[92] "Download Universal SafetyNet Fix [Magisk Module]," Aug. 2022, section: Magisk Module. [Online]. Available: https://magiskroot.net/universal-safetynet-fix/ [Cited on page 48.]

[93] "How to Keep Magisk Root after OTA Update? | MagiskRoot," Dec. 2022, section: Rooting. [Online]. Available: https://magiskroot.net/how-to-keep-magisk-root-after-ota-update/ [Cited on page 48.]

[94] "GitHub - RikkaApps/Riru: Inject into zygote process." [Online]. Available: https://github.com/RikkaApps/Riru [Cited on page 49.]

[95] rovo89, "rovo89/XposedBridge," Aug. 2023, original-date: 2012-03-31T12:01:55Z. [Online]. Available: https://github.com/rovo89/XposedBridge [Cited on page 49.]

[96] "Dexmaker," Aug. 2023, original-date: 2014-12-17T07:22:19Z. [Online]. Available: https://github.com/linkedin/dexmaker [Cited on page 49.]

[97] J. Wharton, "Android dalvik Dx Library," Aug. 2023, original-date: 2016-01-26T19:58:39Z. [Online]. Available: https://github.com/JakeWharton/dalvik-dx [Cited on page 49.]

[98] Lody, "SandHook," Aug. 2023, original-date: 2019-01-12T18:32:08Z. [Online]. Available: https://github.com/asLody/SandHook [Cited on page 49.]

[99] jmpews(AKA.zz), "jmpews/Dobby," Aug. 2023, original-date: 2017-08-03T17:51:59Z. [Online]. Available: https://github.com/jmpews/Dobby [Cited on page 49.]

[100] J. Wu, "topjohnwu/Magisk," Aug. 2023, original-date: 2016-09-08T12:42:53Z. [Online]. Available: https://github.com/topjohnwu/Magisk [Cited on page 49.]

[101] "Google Maps Usage Statistics 2023: The Most Important Facts • GITNUX," Aug. 2023, section: Market Data. [Online]. Available: https://blog.gitnux.com/google-maps-usage-statistics/ [Cited on page 62.]

[102] "Most downloaded travel apps worldwide 2022." [Online]. Available: https://www.statista.com/statistics/1229187/most-downloaded-travel-apps-globally/ [Cited on pages 62, 63, and 71.]

[103] D. R. Piranda, D. Z. Sinaga, and E. E. Putri, "ONLINE MARKETING STRATEGY IN FACEBOOK MARKETPLACE AS A DIGITAL MARKETING TOOL," *JOURNAL OF HUMANITIES, SOCIAL SCIENCES AND BUSINESS*, vol. 1, no. 3, pp. 1–8, Mar. 2022, number: 3. [Online]. Available: http://ojs.transpublika.com/index.php/JHSSB/article/view/123 [Cited on page 65.]

[104] "Terms of Location-Based Services | Facebook Help Centre." [Online]. Available: https://www.facebook.com/help/626057554667531 [Cited on page 65.]

[105] M. H. Alkawaz, H. Rajandran, and M. I. Abdullah, "The Impact of Current Relation between Facebook Utilization and E-Stalking towards Users Privacy," in *2020 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, Jun. 2020, pp. 141–147. [Cited on page 66.]

[106] K. Church and R. De Oliveira, "What's up with whatsapp?: comparing mobile instant messaging behaviors with traditional SMS," in *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*. Munich Germany: ACM, Aug. 2013, pp. 352–361. [Online]. Available: https://dl.acm.org/doi/10.1145/2493190.2493225 [Cited on page 67.]

[107] R. Khan, S. Barakat, L. AlAbduljabbar, Y. AlTayash, N. AlMussa, M. AlQattan, and N. S. M. Jamail, "WhatsApp: Cyber Security Risk Management, Governance and Control," in *2022 Fifth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*, Mar. 2022. [Cited on page 68.]

[108] "Mobile browser market share worldwide 2022." [Online]. Available: https://www.statista.com/statistics/263517/market-share-held-by-mobile-internet-browsers-worldwide/ [Cited on page 70.]

[109] D. D. Rathod, "WEB BROWSER FORENSICS: GOOGLE CHROME," *International Journal of Advanced Research in Computer Science*, 2017. [Cited on page 70.]

[110] J. Skov, "New urban mobility ecosystem," 2017. [Cited on page 71.]

[111] "microG Project." [Online]. Available: https://microg.org/ [Cited on pages 74 and 75.]

[112] W. Luo, S. Xu, and X. Jiang, "Real-time detection and prevention of android sms permission abuses," in *Proceedings of the First International Workshop on*

Security in Embedded Systems and Smartphones, ser. SESP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 11–18. [Online]. Available: https://doi.org/10.1145/2484417.2484422 [Cited on page 74.]

[113] S. Ramachandran, A. Dimitri, M. Galinium, M. Tahir, I. V. Ananth, C. H. Schunck, and M. Talamo, "Understanding and granting android permissions: A user survey," in *2017 International Carnahan Conference on Security Technology (ICCST)*, 2017, pp. 1–6. [Cited on page 74.]

[114] "LSPosed Framework," Aug. 2023, original-date: 2021-02-02T07:22:47Z. [Online]. Available: https://github.com/Magisk-Modules-Repo/riru_lsposed [Cited on page 76.]

[115] "VirtualXposed/README.md at vxp · android-hacker/VirtualXposed." [Online]. Available: https://github.com/android-hacker/VirtualXposed/blob/vxp/README.md [Cited on page 77.]