UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

DOTTORATO DI RICERCA IN MATEMATICA E INFORMATICA XXXII CICLO

GABRIELLA VERGA

Identification of Privacy Risks for Android Users and Effective Protection Mechanisms

TESI DI DOTTORATO DI RICERCA

Tutor: Chiar.mo Prof. Emiliano Tramontana

Anno Accademico 2019 - 2020

To my husband and my parents

...

I would like to thank my supervisor, Prof. Tramontana, for the help provided to me in all these years, the great knowledge that he has given me and for the availability and precision shown to me during the whole drafting period. Without his, this research work would not have come to life!

Abstract

During the last few years, smartphones have seen a marked increase in popularity, due to its many attractive features such as GPS functionality and apps such as e-mail clients, address book, and many more. The world's most popular mobile Operating System is Android, consequently Android devices are at the center of much research which discuss their points of strengths and weaknesses. An important tool of Android smartphones is Application Programming Interface (API); it is a combination of benefits, meaning they allow to create user services, and risks, especially for user privacy. Taking advantage of the API, it is possible to make data travel from our device and third parties via network and vice versa.

This thesis analyzes the data flow on smartphones with Android Operating System. We present, through the APIs and the network, how the user's information leaves the mobile device, putting the user's privacy at risk and how the information invades the device influencing user behavior.

We deal with various techniques to create useful services for citizens through user profiling and how similar procedures can be used to create attack scenarios to steal personal data, highlighting users misunderstandings about the use of their data by applications.

Moreover advanced tools were presented to guide the user in using the mobile device and understanding the behavior of apps installed on the device, protecting their security and avoiding the loss of sensitive information.

Each proposed solution has been tested with multiple real datasets containing data taken from mobile devices, demonstrating the potential of methods proposed.

Contents

Chapter 1

Introduction

Today's users spend more time using their electronic devices, such as PCs, tablets and smartphones. A *smartphone* (or smart phone or touch phone or mobile device) is a portable, battery-powered device that combines the features of a mobile phone with those of processing and transmitting data typical of the world of personal computers; it also uses sensors to determine the location (i.e. GPS position) and to acquire other elements of the environment surrounding the user.

In the last few years, smartphones have seen a marked increase in popularity, so much that users prefer to use the mobile device in lieu of a portable device. Since 2010, the mobile phone is the new personal computer, the desktop computer is not going away, but the smartphone market has grown fast [1]. Smart phones are used as computers by more people and for more purposes and they are generally cheaper than computers, more convenient because of their portability, and often more useful with the context provided by geolocation.

Since 2011, smartphone sales were expected to exceed desktop PCs[1]. This migration to smartphone is also due to its many attractive features which tend to increase and improve every day, such as GPS related functionalities and apps such as e-mail clients, address book, and many more [2]. The mobile phone initially was an essential tool, within the reach of a few, whose possession fulfilled the function of constantly making a privileged number of users "socially engaged and important" traceable in real time, today, smartphone has become an object of entertainment daily; users use it to follow the communities of different social networks, as well as to share photos, chats, links, voice notes. Soon the smatphone has responded and fueled the common need to be close, overcoming the boundaries of space and time, profoundly transforming the possibilities of daily relationships, favoring the possibility of increasing opportunities for intimacy and, sometimes, even those of violation of freedom, personal spaces and *private information*.

[1] L. Snol. More smartphones than desktop PCs by 2011. http://www.pcworld.com/article/171380/more_smartphones_than_desktop_pcs_by_2011.html.

Among the devices on the market, Android operating system (OS)[2] stands out which powers billions of devices ranging from phones to watches, tablets, TVs, and more. In 2014, Android has become the most popular operating system, surpassing Windows Phone, iOS and Firefox OS [3]. The success of Android OS is partly due to open architecture and the popularity of its application programming interface (API) in the developer community [4, 5]. Android OS offers an ample amount of apps, available on the official app market. Among the many features, applications also offer services to users, various of them take advantage of the GPS coordinates; given the technologies tracing one's geographical position, it becomes increasingly easier to acquire information relating to users' GPS coordinates in real time. This availability has triggered several studies based on user positioning, such as the analysis of the flows of people in the cities [6], or the prediction of people movements [7], or the improvement of services that identify the points of interest [8, 9].

Generally, apps provide enhanced services which can be discovered simply by searching some keywords and apps can be easily installed on a smartphone according to needs and preferences. Once installed, they have access to several *resources* available on the smartphone, by resorting to appropriate APIs [10]. Resources are the like of *sensors* for reading GPS user coordinates, amount of light, etc., *file system* for reading and storing data, user *contacts*, *microphone*, etc. In an Android device, resources (like GPS component, to read the user coordinates) are managed under Android OS dangerous level permissions [11]. Moreover, several other resources, such as bluetooth, are managed under Android OS normal level permissions. While for dangerous level permissions users are asked to grant the app access to resources, normal level permissions are automatically granted to an app using the related resources.

Despite providing useful features, sometimes such apps behave as a gateway for sensitive data to flow over the network; the user can experience a loss of control over her own data, since it becomes unknown whether an app, or a remote side, uses her personal data, such as e.g. location, contacts, etc. [12, 13]. In fact, although the Android OS permission system was created to protect the user and his privacy, it is vulnerable to some attacks [14, 15]. Such attacks can create several inconveniences for users, including loss of private information. Smartphone attack types include cellular networks, Bluetooth, the Internet (via WiFi or 3G/4G network access), USB, and other peripherals [16], some of these attacks come from mobile applications. Indeed, today's society is in constant movement and change, thanks to mobile technology, cloud or Internet of things, nevertheless, a system (comprising Android OS and all the installed apps) so capable of acting and

[2] www.android.com

reacting becomes an attack vehicle that can allow someone to:

- *acquire data improperly*: access the smartphone to acquire sensitive information such as geographic location, telephone contacts and more. The process usually takes place by inducing the victim user to download an app that is able to extract information and send it to a server through a network connection. Even if the user cannot see an attack or malfunctions on the system, he undoubtedly has a clear loss of control over what the data is, who knows his data, where data is and who does what with data.

- *enter data into the system*: send information to the user in order to influence his choices or thoughts. The reading of advise (for example through recommendation systems or targeted advertising) or false information (for example the so-called fakenews) can influence the restaurant where to have lunch, the detergent to buy or worse influence political choices.

It is commonly accepted that, to provide services, the smartphone asks user's private information, which creates smartphone security problems both in the case of theft, leaks of data or data sniffing, and increasing problems due to the ample time spent connected to the web world, thus reinforcing the need to have a "SECURE SYSTEM" to prevent illicit operations [17].

1.1 Network impact on Android devices privacy

The interest in smartphones grows with the desire to remain always connected to stay in close contact to the world, to see the latest videos posted by some other users, the latest news on the world or to communicate with friends via social networks and more. Nowadays, the risks connected to smartphones and their applications derive essentially from the fact that our mobile phones are constantly connected and consequently localized, and that the large number of data and information contained in them, from phone books to diaries, from photos to notes, can be known, processed, stored, used by third parties of whom we have no awareness or control (see Figure 1.1).

The INTERNET PERMISSION is the needed related normal permission for using the network. When this permission is used together with others, generally dangerous level permissions, allows sharing information between our device and third parties via internet. There is a set of information that reaches the user (i.e. advertising, recommendation systems, etc.) and a set of information concerning the user himself is transmitted out of the device (i.e. GPS position, telephone contacts, voice messages, etc), exploiting to the

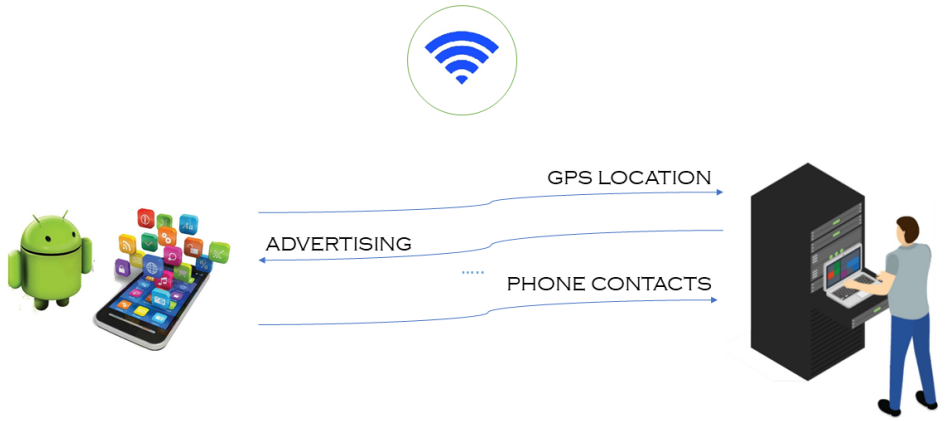

Figure 1.1: The connection between our device (on the left) and third parties (on the right) via the connection network. The arrows indicate the direction in which the information is traveling.

network. The user is often not aware of the leakage of his data, consequently, a set of issues are raised: what data are sent / received; to whom data arrive; how these data are used; how and when user privacy is preserved.

All these problems arise when data cross the boundary between our device and the network connection. Thus, new needs are born in terms of security for the protection of the user connected and the related risks. Among the major risks involved there are: risks of leaking sensitive information and the risks to receive unsolicited information that can for example alter our state of thought or simply become useless because not complying with our needs.

In brief, users who are continuously connected to the network can take advantage of different services dedicated to them, often useful for the user but at the same time risky: leak of data, which are continually sent without having full knowledge of it; absorb non-relevant information, as the user may receive targeted advertising or unsuitable news for his interests. Consequently, it is important for the user to be fully aware of the boundary between his own device and the outside world to actually know which data enters his device and which data leaves it.

1.2 User knowledge and security problems

It seems that there is still a long way to go to increase the awareness of European citizens in terms of personal data protection, especially regarding the use of apps. According to

the latest data released by Eurostat on the occasion of Data Privacy Day[3], in fact, 28% of the inhabitants of one of the EU member countries have never limited apps from access into personal information. Even worse, seven out of a hundred people do not even know that it is possible to carry out such limitation of accesses on their smart phones. Yet, 75% of Europeans interviewed (between 16 and 74 years of age) use a smartphone on a daily basis for personal purposes.

To facilitate users, Android developers have introduced the concept of permissions, so that the user can choose which authorization to grant to each application. But, yet, several studies have shown that users pay limited attention to permission requests and have a limited comprehension on permissions. Surveys have reported that less than 20% of users paid attention to permissions during installation, and only 3% can remember them [18, 19]. This leads to the conclusion that permissions are not properly used by the user and, consequently, the user often confirms permissions he does not understand or remember. This leads to having applications with too many permissions granted and leads to canceling the usefulness of the system of permissions because the user is not able to understand them. In addition, some applications (like twin applications) can easily trick users, i.e. by simulating the behavior of a known application they can extract sensitive data from the user.

So, the first problem related to user security, revolves around the *naivety of users and their lightness in data sharing*.

Another problem related to user security is *the management of permissions*. For protecting the user, several resources of the Android OS are shielded by *permissions*, therefore apps have to declare the needed permissions before accessing the related resource [10]. There are two main permission levels *normal* and *dangerous*: normal-level permissions are accepted by the user automatically upon installation of the app; dangerous level permissions are asked to the user, who can grant them by means of a dialog window on a per-use basis at runtime. However, the first level, normal level permission, leaves the user unaware of which resources the app is accessing and when the app is using them; such a category includes the use of Internet, access of network state, use of phone vibration, etc. For the second level, dangerous permission, the user is given an alert when a corresponding resource is used (such an alert can be disabled by the user); such a category includes the use of the camera, contacts, location, microphone, phone, storage. For both levels, whether the app uses gathered data only locally or send them remotely is unknown. Usually, apps declare some set of normal permissions together with some set of dangerous permissions. It is mostly their combined use that can lead to some privacy

[3] https://ec.europa.eu/eurostat/web/products-eurostat-news/-/EDN-20190128-1?inheritRedirect=true&redirect=%2Feurostat%2F

breach. Let us suppose that dangerous permissions are declared to enable the app access some sensitive data, as the user contacts, and normal permissions are declared to access internet or perform background activities. Now, the combined use of both permissions would allow an app to send sensitive data over the network, even while the user is not interacting with the app. Some studies have reported such effects, such as e.g. malicious apps that declare dangerous permissions: WRITE SMS, SEND SMS, FINE LOCATION; and declare normal permission INTERNET [20]. Normal permissions could allow an app to violate the privacy of the users, and increase the risk of data loss. E.g. a user might know that her GPS coordinates are read if she navigates in a map, however does not know that the app could run in background and send updated data read from GPS, continuously. This is because Android OS allows an app to run in the background and connect to the Internet. Then, as soon as the user grants access to some data (e.g. her position or contacts) to the app, such data may flow over the network, without letting the user know about it. For such an app, normal permissions FOREGROUND SERVICE and INTERNET have to be declared, in addition to dangerous permission LOCATION.

Once granted, dangerous permissions would allow an app to extract user sensitive information, which could be then used to harm the user. E.g. dangerous permissions are asked by an app when it needs to read GPS coordinates, or the phone contacts, etc. Moreover, after consent has been granted, data could be read or sent over the network for the whole duration of the app execution, without restriction or temporal limitation. This might lead to a loss of control on how data are used. The problem of dangerous permissions is hidden in the unsupervised handling of transmission of data from the device to a remote server. The user loses control of the data because she does not know when and with what frequency the data are read or forwarded. E.g. if a user opens a map in an app, this app asks the permission to access the GPS data on the device, however the data coordinates can be read and sent over the network even after leaving the map dialog window, without user awareness. In addition, if the app accesses other data, the risk of revealing sensitive data is higher due to the possible correlation between different parts of data.

Finally, it is interesting to highlight the *influence of information on the reader*. Users are very influenced by the news they receive, for this reason among the hot topics we also find the study of human thoughts. The interpretation of human thoughts is a very important field, because it allows us to achieve a variety of results, shown by several widely used applications [21]. E.g. many companies need to analyze opinions that users have expressed on some products. Thanks to the widespread use of the smartphones, users share information and thoughts on the net with greater ease and simplicity, so, this field has gained more and more attention and thousands of people tend to have collaborative

and entertaining web-based relationships (e.g. being members of internet social networks and forums) as an integral part of their life. Hence, many Internet pages reflect people's thoughts, feelings and opinions.

For all the reasons listed above, it is important to protect the user by analyzing the data entering and leaving the device and guiding the user towards a greater awareness of their own risks.

1.3 The proposed approach

Nowadays, user privacy is widely discussed in various fields because many companies focus their business on the collection and combination of data (especially sensitive data). Smartphones are one of the most used means for data mining, given their widespread use and the large quantity of data they offer.

This thesis discusses the data management on Android smartphones and the permissions system that is a combination of risks and benefits [20]; the work offers an analysis of the users risks and it focuses on the border between the device and the network connection to analyze and study what is spread around, which data originates from the device, and what information comes from third parties. The study is widely based on data analysis and their flow and highlights how it is possible to create:

- *scenarios of attack*: the APIs allow users to create new attacks to extract user sensitive data. In several studies, the use of normal and dangerous permissions have been linked to possible attacks and loss of sensitive data [22, 23];

- *user profiling*: through the use of user profiling it is possible to create services dedicated to users that help people in everyday life. There are many services illustrated in the literature such as: extracting interesting positions and travel sequences [24], the process of aligning a sequence GPS with the road network on a digital map, useful for pre-processing step for many applications, such as traffic flow analysis [25] or the analysis of the attractiveness of the shopping center to benefit traffic management and urban planning [26].

- *tools of protection*: both services and attacks create a leak of personal data, so it is important to create a protective shield for the user. In literature, there are several frameworks that identify a possible attack on the device including Andromaly [27], a framework that realizes a host-based malware detection system that continuously monitors various features and events obtained from the mobile device and then applies some machine learning based detectors to classify collected data as normal

(benign) or abnormal (harmful). Among these, many methods are based on permission monitoring, e.g. AndroidLeak [12] that automatically detects potential privacy leaks in apps based on maps of permissions and other apps data.

In this thesis, we will illustrate several ways to create services for citizens from a data collection based on user profiling and we will investigate attack scenarios for users of Android devices, giving some practical examples of the problems related to the gathering of user location.

To assist the user in containing his data, we will present two elaborate defense mechanism capable of alerting the user to possible threats, i.e. loss of data. Two approaches have been presented that warn the user in real time to alert them of the operations performed by the applications. By means of these works, users can understand whether a newly installed app works in the background by sending large amounts of data, or how an application works, that is every time data is taken, what data is taken and how often. These new approaches are a good contribution to user privacy.

Finally, these new approaches have been evaluated with real data and applications; consequently, due to the large amount of data that can be taken from the devices, filtering and parallel programming techniques were used for analysing them in order to gain some knowledge needed to protect the user. Pre-processing of data allows better and more reliable results to be obtained and also reduces the response times of algorithms for data management.

1.4 Thesis structure

Thesis is structured as follows. Chapter 2 discusses the required fundamentals for the work described in further chapters; the chapter presents an overview of Android and its APIs, useful for extracting information from the device, together with attacks on Android OS, data leak and users privacy risks. Following, we discuss the advantages and disadvantages of user profiling and the methods present in the literature to protect the user.

Chapter 3 deals with how to take advantage of data extracted from smartphone. The Chapter discusses user services and attacks created through the API. It introduces benevolent tools with the aim of improving services for citizens, in particular we present two approaches, the first with the several intent, for example to enhance public transportation or transportation planning and the second with the intention of helping the user to find Points Of Interest (POIs). Such data can be used, also, to attack the user, in fact, subsequently a type of attack is presented for Android devices.

Having introduced how the data can be extrapolated, the Chapter 4 explains how the data can be entered into the device and its effects for the user. POI recommendation is one of such services, which is to recommend places where users have not visited before; the Chapter discusses of a service based of POI recommendation. The recommendation is reinforced by the opinion of other users who have already visited the place, so, such Chapter introduces the concept of Sentiment Analysis. Sentiment analysis and opinion mining is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions from written language [28]. The chapter ends with a targeted advertising attack which silently tracks the user position.

Chapter 5 discusses countermeasures to protect the user in order to shield her from attacks or data loss. It introduces two innovation techniques and, furthermore, it explains how to apply the second method illustrated for the attacks discussed in Chapters 3 and 4. Finally, Chapter 6 gives conclusions for this thesis.

1.5 Papers 2016 - 2019

The work presented in this thesis is partially based on these co-authored papers:

- C. Cavallaro, G. Verga, E. Tramontana, O. Muscato: *Eliciting Cities Points of Interest from People Movements and Suggesting Effective Sightseeing Itineraries*. **Submitted** to the IOS Press Intelligenza Artificiale the International Journal of the AIxIA, 2019 (in review).

- G. Verga, A. Fornaia, S. Calcagno, E. Tramontana: *Yet another Way to Unknowingly Gather People Coordinates and its Countermeasures*. In **Proceedings** on the 12th International Conference on Internet and Distributed Computing Systems (IDCS). Springer, LNCS 11874.

- G. Verga, A. Fornaia, S. Calcagno, E. Tramontana: *Smart Cities and Open WiFis: when Android OS Permissions Cease to Protect Privacy*. In **Proceedings** on the 12th International Conference on Internet and Distributed Computing Systems (IDCS). Springer, LNCS 11874.

- C. Cavallaro, G. Verga, E. Tramontana, O. Muscato: *Multi-Agent Architecture for Point of Interest Detection and Recommendation*. In **Proceedings** of the 20th Workshop From Object to Agents (WOA), 2019.

- E. Tramontana and G. Verga: *Mitigating Privacy-related Risks for Android Users*. In **Proceedings** of the 28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2019.

- E. Tramontana and G. Verga: *Get Spatio-Temporal Flows from GPS Data*. In **Proceedings** of the 4th IEEE International Conference on Smart Computing (SMARTCOMP), 2018.

- A. Di Stefano, A. Fornaia, E. Tramontana, G. Verga: *Detecting Android Malware According to Observations on User Activities*. In **Proceedings** of the 27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2018.

- C. Napoli, E. Tramontana, G. Verga, G: *Extracting location names from unstructured italian texts using grammar rules and mapreduce*. In **Proceedings of International Conference** on Information and Software Technologies (ICIST). Springer, 2016.

Chapter 2

Background And Related Works

Android is the world's most popular mobile Operating System (OS), powering billions of devices [3]. Users of Android OS devices can download an app from the market and install it on their device. Sometimes such apps behave as a gateway for sensitive data to flow over the network and user can experience a loss of control over her own data, since it becomes unknown whether an app, or a remote side, uses her personal data, such as e.g. location, contacts, etc. [12, 13]. Consequently, through of applications, users are easy prey for attacks, for example stealing personal data but at the same time using the services of apps have better navigation and profiling. We give the definition of attack and service.

An *attack* (or cyber attack) is any maneuver, used by individuals or even state organizations, which affects information systems, infrastructures, computer networks and/or personal electronic devices through malicious acts, generally coming from an anonymous source, aimed at the theft, alteration or destruction of specific targets in violation of susceptible systems. Generally, the greatest risk is the theft of confidential and personal data. Given the high growth of smartphones, there is an increase in cyber attacks on security on smartphones. A Kaspersy report sounds the alarm bell: virus-infected smartphones are growing rapidly, in fact, between the end of 2017 and the end of 2018, they have doubled.

Furthermore, a *service* is an application created with the aim of facilitating users using the smartphone; it can perform long-running operations in the background, and it generally provide an user interface. For example, starting from GPS data, services are created for smart city solutions. The term "smart city" is used to describe applications of complex information systems to integrate the operation of urban infrastructure and services such as buildings, transportation, electrical and water distribution, and public safety [29].

The popularity of smartphones as well as problems arising from the lack of security have been widely documented [14, 15]; to provide services, the device requires user's private information, which creates smartphone security problems both in the case of theft, loss or data sniffing, and increasing problems due to the ample time spent connected to

the web world. It reinforces the need of a 'secure system' to prevent illicit operations [17] and the need to improve the security in such devices has become paramount.

In this chapter we discuss everything required to guide the reader toward the work of this thesis. In Section 2.1 we introduce the Android smartphone world. The section explains how to access resources through the APIs, introduces the security mechanisms created by the Android developers, namely the permission system and discusses about problems related to user security. Here, a list of well-known APIs and a list of the most used and sensitive permissions to privacy are shown.

In Section 2.2 we introduce the Android vulnerabilities analyzed during the aforementioned thesis work. The main topic is the attack on the devices Android with the purpose of damaging the user and steal personal data. The subcategories exposed are malware and Man-in-The-Middle attacks.

In Section 2.3 we explore the concept of user profiling. The combination of data and their correlation is a source of great gain as it allows an actor to acquire useful information in the market. Thanks to the analysis of large amounts of users data it is possible to create more effective studies and test them with real data. In this section, we discuss how to use this data for services based on user profiling. We offer an introduction of some data management techniques (filtering, parallelism) and illustrates datasets used in our tests together with the related works in the literature. Datasets have data that come mainly from smartphones, gathered via some APIs.

In Section 2.4 we discuss of tools to protect the user, in particular we discuss techniques to detect data loss and others to detect presence of malware.

2.1 Android Operating System

Since 2012, Android OS popularity has surpassed Symbian and iOS, being installed on over half (52.5%) of all smartphones shipped [30]. In 2014, Android has surpassed Windows Phone, iOS and Firefox OS [3]. Since April 2017, Android is the most widespread mobile OS in the world, with a market share of 62.94% of the total, followed by iOS with 33.9%[1].

Android OS has been developed by Google Inc. and is based on Linux OS kernel; it is considered an embedded Linux distribution, since almost all GNU utilities are replaced by Java counterparts.

Currently, most Android apps are being developed in Java, use the Android Software Development Kit (SDK), and are compiled to Java bytecode. They are then converted

[1] www.supinfo.com/articles/single/4760-history-of-android-evolution

from Java Virtual Machine-compatible class files to Dalvik-compatible, or DEX (Dalvik Executable), files that run on Android Dalvik Virtual Machine (DVM). During the Android app build process, DEX files are bundled into an Android Application Package (APK) file. While not all Java bytecode can be translated to DEX, many Java libraries are compatible with Android OS.

Security problems on Android are well known; smartphone security has matured considerably, and the literature has reported several papers, including an introduction to Android security [31], and smartphone security surveys [32]. One of the security mechanisms embedded in Android OS is the permission system. Android library APIs that give apps the possibility to access some critical resource, such as microphone, contacts, etc. are managed by the permission system.

2.1.1 API and Permissions

The extrapolation of data creates new problems in terms of user security and privacy [33], partially due to the misunderstanding of security policies by users [18, 13].

Android library APIs [34] that give apps the possibility to access device resources; the following provides the list of some APIs, i.e. classes and methods, that can be used to extract useful information from an Android device with a description of such Android APIs and we show the relationship between the described APIs and the corresponding Android class (see Table 2.1).

- *LocationManager* is a class providing access to the system location services. Its method *getLastKnownLocation(String provider)* returns a Location indicating the data from the last known location point obtained from the given provider.

- *WifiInfo* describes the state of any WiFi connection that is active or is in the process of being set up. The *getSSID()* method returns the service set identifier (SSID) of the current 802.11 network.

- *WifiManager* class provides the primary entry point for managing all aspects of WiFi connectivity. Method *getWifiState()* provides the WiFi enabled state, and *getConnectionInfo()* returns dynamic information about the current WiFi connection, if any is active.

- *TrafficStats* class provides network traffic statistics. These statistics include bytes transmitted and received and network packets transmitted and received, over all

interfaces, over the mobile interface, and on a per-UID basis. Methods *getMobileRxBytes()* and *getMobileTxBytes()* return the number of bytes received and transmitted, respectively, across mobile networks since device boot.

- *PackageManager* class lets us retrieve various kinds of information related to the application packages that are currently installed on the device. Method *getApplicationInfo(String packageName, int flags)* gives information about a certain package/application, and method *getInstalledApplications(int flags)* returns a list of all application packages that are installed on the device.

- *NetworkInfo* describes the status of a network interface. Method *getType()* reports the type of network to which it pertains, and method *isConnected()* indicates whether network connectivity exists and whether it is possible to establish connections and pass data.

- *ConnectivityManager* monitors network connections (WiFi, GPRS, UMTS, etc.). Method *getNetworkInfo(Network network)* returns the connection status about a given network, whereas method *getAllNetworks()* returns an array of all networks currently tracked by the framework.

- *BatteryManager* analyses the battery level. Method *extra-level()* gives a value which is an integer representing the current battery level, from 0 to extra-scale which contains the maximum battery level.

- *RunningAppProcessInfo* gives information about a running process. E.g. *processName()* returns the name of the process that this object is associated with.

In Android OS, resources are protected by a mechanism based on permissions, which has been widely discussed in the literature [35, 36], and constitutes a weak point suitable for attacks and data leak.

Android library APIs [34] that give apps the possibility to access some critical resource, such as microphone, contacts, etc. have been assigned a permission level. Then, apps wishing to call such APIs have to declare the corresponding permission in a file, bundled into the APK, dubbed *AndroidManifest.xml*.

Android developers have grouped permissions into different levels of protection [11], i.e. normal and dangerous, mainly[2]. The levels refer to the intended use of a permission, as well as the consequences of using the permission.

[2]Another level available is Signature, and if used the app key has to match the platform key to access core platform packages.

Table 2.1: Android APIs

Package	Class	Level	Since version
android.location	LocationManager	1	1.0
android.net.wifi	WifiInfo	1	1.0
android.net.wifi	WifiManager	1	1.0
android.net	TrafficStatNow	1	1.0
android.net	NetworkInfo	1	1.0
android.net	ConnectivityManager	1	1.0
android.os	BatteryManager	1	1.0
android.app	RunningAppProcessInfo	3	1.5
android.content.pm	ApplicationInfo	1	1.0
android.net	Network	21	5.0

NORMAL permission level is for APIs considered mostly harmless for the privacy of users, or for the operation of other apps. Apps have to declare such permissions and Android OS then grants them immediately upon installation of an app. Essentially, the user is unaware of the declaration of such permissions. An example of APIs under normal permission level is the one for setting the time zone. A selection of normal permissions is shown in Table 2.2.

DANGEROUS permission level is for APIs that affect users private data, or could potentially affect phone calls, or the operation of other apps. Such permissions, similarly to normal ones, must be declared by an app. Since version 6.0 of Android OS (released on October 5, 2015), the user has to grant access to the app at runtime, before the app can use the corresponding APIs [35]. Figure 2.1 shows how requests are performed at runtime. The second time the request is performed, the user can choose to grant access permanently. An example is the permission to read user contact data [3]. The whole list of dangerous permissions is given in Table 2.3.

When executing an Activity or code Fragment that uses an Android library API the corresponding permission has to be declared. E.g. WakeLock Android library API class lets an app force the device stay on; any app using WakeLock has to declare the android.permission.WAKE_LOCK normal permission before using the methods available from class android.os.PowerManager.WakeLock.

The above described permission levels and mechanisms are not meant to check, nor trace, control flows or data flows within an app at runtime [20]. I.e., once an app has gained a set of dangerous permissions, the moments in which an app is using APIs under such a set of permissions is unknown. E.g. the events or user inputs that trigger the use

[3] developer.android.com/guide/topics/permissions/overview

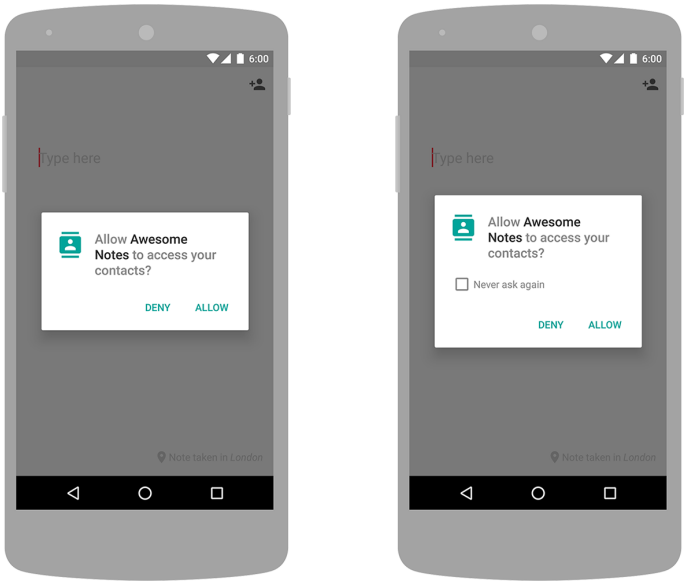

Figure 2.1: Initial permission dialog (left) and secondary permission request with option to turn off further requests (right).

of the microphone for an app is unknown to the device owner. Therefore, an app having asked permission to use the microphone and internet could send samples of recorded sound any time during its execution, even while in background. The user remains unaware of it, however it poses a serious risk to privacy.

In the next section we illustrate the problems of user privacy due to permission management, together with the interventions of such problems present in the literature.

2.1.2 Privacy risks related to permission management

One of the problems of Android mobile devices is related to the risk of private data flowing from the device to an unknown remote actor via API. By systematically analysing the Android APIs code, a previous research study has shown the APIs that are unprotected by any permission, and using them allows an app to discover sensitive data, such as device id, setting data, power modes, etc., as well as perform actions such as set volume, change alarm tones, etc. [37]; even when APIs are protected by permissions, some unexpected app behaviour cannot be averted. E.g. an app could turn on the microphone, having previously obtained the needed permission, when the user is checking the list of messages or when the app runs in background, this leads to risks for users [38]. Otherwise, while the user could be aware that an app is reading her contacts (or GPS coordinates) and has allowed the app to access them, she is unaware that her data are transmitted remotely. E.g. the ACCESS_WIFI_STATE permission allows applications to access information about Wi-Fi networks and is a normal permission level. In literature, a well-known example

Permissions	Description
ACCESS_LOCATION_EXTRA_COMMANDS	Allows an application to access extra location provider commands
ACCESS_NETWORK_STATE	Allow applications to access information about networks
ACCESS_NOTIFICATION_POLICY	Marker permission for applications that wish to access notification policy
ACCESS_WIFI_STATE	Allow applications to access information about WiFi networks
BLUETOOTH	Allows applications to connect to paired bluetooth devices
BLUETOOTH_ADMIN	Allows applications to discover and pair bluetooth devices
BROADCAST_STICKY	Allows an application to broadcast sticky intents
CHANGE_NETWORK_STATE	Allows applications to change network connectivity state
CHANGE_WIFI_MULTICAST_STATE	Allows applications to enter Wi-Fi Multicast mode
CHANGE_WIFI_STATE	Allows applications to change WiFi connectivity state
DISABLE_KEYGUARD	Allows applications to disable the keyguard if it is not secure
EXPAND_STATUS_BAR	Allows an application to expand or collapse the status bar
FOREGROUND_SERVICE	Allows a regular application to use Service.startForeground
GET_PACKAGE_SIZE	Allows an application to find out the space used by any package
INSTALL_SHORTCUT	Allows an application to install a shortcut in Launcher
INTERNET	Allows applications to open network sockets
KILL_BACKGROUND_PROCESSES	Allows an application to call ActivityManager.killBackgroundProcesses(String)
MODIFY_AUDIO_SETTINGS	Allows an application to modify global audio settings
NFC	Allows applications to perform I/O operations over NFC
READ_SYNC_SETTINGS	Allows applications to read the sync settings
READ_SYNC_STATS	Allows applications to read the sync stats
RECEIVE_BOOT_COMPLETED	Allows an application to receive the Intent that is broadcast after the system finishes booting
REORDER_TASKS	Allows an application to change the Z-order of tasks
REQUEST_INSTALL_PACKAGES	Allows an application to request installing packages
SET_ALARM	Allows an application to broadcast an Intent to set an alarm for the user
SET_TIME_ZONE	Allows applications to set the system time zone
SET_WALLPAPER	Allows applications to set the wallpaper
SET_WALLPAPER_HINTS	Allows applications to set the wallpaper hints
TRANSMIT_IR	Allows using the device's IR transmitter, if available
USE_BIOMETRIC	Allows an app to use device supported biometric modalities
VIBRATE	Allows access to the vibrator
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming
WRITE_SYNC_SETTINGS	Allows applications to write the sync settings

Table 2.2: A selection of normal level permissions

Permissions	Description	Group
READ_CALENDAR	Allows an application to read the user's calendar data	CALENDAR
WRITE_CALENDAR	Allows an application to insert or update calendar data	CALENDAR
READ_CALL_LOG	Allows an application to read the user's call log	CALL_LOG
WRITE_CALL_LOG	Allows an application to write (but not read) the user's call log data	CALL_LOG
PROCESS_OUTGOING_CALLS	Allows an application to see the number being dialed during an outgoing call with the option to redirect the call to a different number or abort the call altogether	CALL_LOG
CAMERA	Required to be able to access the camera device	CAMERA
READ_CONTACTS	Allows an application to read the user's contacts data	CONTACTS
WRITE_CONTACTS	Allows an application to write the user's contacts data	CONTACTS
GET_ACCOUNTS	Allows access to the list of accounts in the Accounts Service	CONTACTS
ACCESS_FINE_LOCATION	Allows an app to access precise location	LOCATION
ACCESS_COARSE_LOCATION	Allows an app to access approximate location	LOCATION
RECORD_AUDIO	Allows an application to record audio	MICROPHONE
READ_PHONE_STATE	Allows read only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device	PHONE
READ_PHONE_NUMBERS	Allows read access to the device's phone number(s). This is a subset of the capabilities granted by READ_PHONE_STATE but is exposed to instant applications	PHONE
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call	PHONE
ANSWER_PHONE_CALLS	Allows the app to answer an incoming phone call	PHONE
ADD_VOICEMAIL	Allows an application to add voicemails into the system	PHONE
USE_SIP	Allows an application to use SIP service	PHONE
BODY SENSORS	Allows an application to access data from sensors that the user uses to measure what is happening inside his/her body, such as heart rate	SENSORS
SEND_SMS	Allows an application to send SMS messages	SMS
RECEIVE_SMS	Allows an application to receive SMS messages	SMS
READ_SMS	Allows an application to read SMS messages	SMS
RECEIVE_WAP_PUSH	Allows an application to receive WAP push messages	SMS
RECEIVE_MMS	Allows an application to monitor incoming MMS messages	SMS
READ_EXTERNAL_STORAGE	Allows an application to read from external storage	STORAGE
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage	STORAGE

Table 2.3: The whole dangerous permissions list

is the use of such permission to extrapolate the user's physical position without them noticing [39]. Moreover, apps often gain more privileges than strictly needed, and the several resources used can be combined in a way to breach privacy [40, 20].

Thus, some weaknesses have been observed on Android OS permission design and mechanisms and in literature works has proposed to analyse apps in order to check whether sensitive data sources are connected to unwanted sinks [41, 42].

In such a context, the most discussed permissions are the *dangerous-level permissions* followed by their companions *normal-level permissions*. Despite the apparent innocence of using normal-level permissions, in several studies, the use of these has been linked to possible attacks and loss of sensitive data [38, 43].

Then, Android has several weaknesses that put the user at risk, among these is permission management as it is sometimes possible to collect sensitive data without the user being aware of it, that is exploiting normal level permissions.

Several techniques have been developed to protect the user's privacy. In [44, 45, 46] the privacy breaches are tackled with cryptographic systems, which can reduce the risk for the user. In other research works, e.g. in [47], each user first disguises her private data, and then sends it to the data collector. Therefore, a Randomized Perturbation (RP) technique is used to disguise private data [48]. Moreover, anonymisation techniques can be used, however these introduce some attack problem, making datasets not very useful [49, 50].

The various techniques alone are not able to protect the user, but they can be a good remedy to mitigate the user's risks, as we will see in the next chapters.

2.2 Vulnerability of Android

The exponentially growing number of Android apps, unofficial app developers and security vulnerabilities existing in the Android OS encourage malicious to create attacks on smartphones to steal private information from users or to damage app markets and the reputation of developers. Since the Android OS is an open source platform that allows the installation of third-party market apps, it is possible installs Android malware able to control of the device, steal private information from users, consume excessive battery power and turn the device in a zombie machine. Furthermore, Android security is built upon a permission-based system which is widely criticized for its coarse-grained control of application permissions and the inefficient permission management, by developers, marketers and end-users [51].

Recent studies have shown that the mobile app market is hosting some apps that are malicious and vulnerable, and putting at risk millions of devices. This is more evident

on Android markets, where in some cases apps have been found infected with malware and spywares and, in other case, attacks come from external sources, such as evil twin of WiFi networks (such attacks can be in the specific Man-in-The-Middle attacks). In evil twin attack, an adversary clones an access point for malicious purposes including malware injection or identity theft [52].

There are many (malicious) applications work by exploiting the users private and monetized information stored in a users smartphone and among these, nearly half of Android malware are multi-functional Trojans that steal personal data stored on the users phone[4]. In addition, in order to extract sensitive information, there are attacks from external sources, among them stand out the attacks Man-in-The-Middle [53].

In this section, we discuss attacks on Android devices; we analyze the concept of malware, that is software installed inside the device, and the concept of Man-in-The-Middle that uses external attacks.

2.2.1 Malware

A MALWARE [54, 55] is a generic term for all types of unwanted software (e.g., viruses, worms or trojan horses) that constitutes a serious security threat to users. E.g., it could consist of a snippet of code inside an apparently trusted app, which acts against the user unknowingly. In fact, such a snippet of code could access saved data, as well as gather everything typed on the keyboard, and/or store any action carried out by the user. Finally, malicious code could periodically send acquired data to a server allowing an attacker to analyse and eventually use them for illicit purposes.

2.2.2 Related Work

Smartphone platforms provide application developers with rich capabilities, which can be used to compromise the security and privacy of the device holder and her environment [56]. The work in [56] examines the feasibility of malware development in smartphone platforms by average programmers that have access to the official tools and programming libraries provided by smartphone platforms, including Android OS, BlackBerry OS, Symbian OS, Apple iOS and Windows Mobile 6 OS.

In the literature we find various descriptions of attacks designed for Android mobile devices [57, 58, 14]. Davi et al. discuss permission redelegation attacks on Android [59], they introduce the problem and present an attack on a vulnerable deputy.

[4] Q2 IT evolution threat report.
http://www.securelist.com/en/analysis/204792299/IT_Threat_Evolution_Q2_2013.

Among the older malware, as discussed in [60], there are: 1) DroidKungFu: the first version of DroidKungFu malware was detected by research team in June 2011 and it was considered one of the most sophisticated Android malware at that time. 2) AnserverBot: it was discovered in September 2011 and it is piggybacks on legitimate apps and is being actively distributed among a few third-party Android Markets in China. The malware aggressively exploits several sophisticated techniques to evade detection.

Among of the malware attacks there are (complete list in [61]):

- Bluetooth attacks: attacker could insert contacts or SMS messages, steals victim's data from their devices and can track user's mobile location.
- SMS attacks: attacker can advertise and spread phishing links.
- GPS location attacks: user's current location and movement can be accessed with global positioning system (GPS) hardware.

2.2.3 Man-In-The-Middle

The MAN-IN-THE-MIDDLE (MITM) concept is used to describe the attack that occurs during communication between a consumer and a legitimate organization. Through the MITM, the attacker is able to sniff packets through encrypted communications between two parties [62, 53].

2.2.4 Related Work

Recently, with the growth of smartphone users, the risks to smartphone owners have become victims of MITM attacks. WiFis are at the centre of many attacks on the privacy of users, who rely on devices such as smartphone or tablet [63, 64]. Among various methodologies, attacks based on the enabled phone's WiFi connectivity options stand out, so that the device can be connected to fake WiFi network without an explicit user operation; exploiting so-called Man-in-The-Middle attack [65, 66]. For example, in [67] the WiFi connection enabled option is used to connect the device to fake access points, or in [68] a detection mechanism was created to find an evil twin Access Point attack which steals sensitive data. Most of these types of attacks are exploited when the WiFi in the devices is active, so they no longer have any merit if the WiFi is turned off.

2.3 User profiling

User profiling is the process of collecting information about an user in order to construct their profile. Knowing users is the fundamental starting point for any marketing strategy and thanks to new technologies based on artificial intelligence, it is possible to profile users like never before.

The massive spread of ICT, the Internet and the growing use of social networks have increased and simplified user profiling, this is also due to use of the API with which it is possible to extract various information of the Android device. This information is used to create large datasets with which the developers create services for users with the backstage of have to adapt to the management of millions of data and the use of techniques for reviewing and cleaning the data.

In this section we discuss of concept of big data and how to manage large amounts of data for the analysis of information that can be extracted from mobile devices, which allow user profiling. We see some services dedicated to users which use information of smartphone and we illustrate a set of datasets on the web and take from real devices.

2.3.1 Data characteristics and management

The term of big data is mainly used to describe a huge amount of data that able us to capture, link, collect, store and organize information. Today, big data related to the service provided by Internet companies grow rapidly, as for Google or Facebook services [69]. This growth is also fueled by data sharing via smarpthones.

Big data brings about new opportunities for discovering new knowledge, helps us to gain an in-depth understanding of the hidden values, but introduce several challenges, including how to capture, transfer, store, clean, analyze, filter, search, share, secure, and visualize data [70]. Data management becomes an increasingly demanding job as each device processes data and puts it on the network, facilitating information sharing. Thus, the use of techniques for the management of large amounts of data has become a source of great interest.

In general, big data shall mean the datasets that could not be perceived, acquired, managed, and processed by traditional IT and software/hardware tools within a tolerable time [69]. Because of different concerns, scientific and technological enterprises, research scholars, data analysts, and technical practitioners have different definitions of big data. So, an relevant challenge of big data is how to effectively organize and manage such datasets.

There are many tools for studying and analyzing big data. Two interesting strategies for processing large amounts of data are:

- Redundancy reduction and data compression: generally, there is a high level of redundancy in datasets. Redundancy reduction and data compression is effective to reduce the indirect cost of the entire system on the premise that the potential values of the data are not affected. For example, most data generated by sensor networks are highly redundant, which may be filtered and compressed at orders of magnitude.

- Parallel Processing: by splitting a job in different tasks and executing them simultaneously in parallel, a significant boost in performance can be achieved. Among other models, the most famous is MapReduce [71] and a famous use of the paradigm is in Hadoop MapReduce. The latter is a software framework for easily writing applications that process vast amounts of data; a MapReduce job usually splits the input data-set into independent chunks that are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then given as input to the reduce tasks. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks [71]. Therefore, MapReduce gives developers a paradigm to organise their tasks, and each new application has to be adapted to fit MapReduce's main two steps as follows:

  - Map is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).
  - Reduce takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

Such techniques are a good starting point for data cleaning and to speed up computational computation and therefore have faster responses.

2.3.2 Related Work

In this section we analyze the literature for jobs related to user profiling by creating services that take advantage of information taken from mobile devices. We will introduce the concept of GPS user trajectories analysis and the study of Points Of Interest (POIs), offering new services for citizens as recommendation systems based on POIs.

GPS trajectories analysis: researchers have ventured into several studies concerning the analysis of user trajectories. This interest is driven by the possibilities it offers for marketing and the many services that can be offered to users.

In the literature, we can find a lot of research that analysed the information to extrapolate new knowledge. Since 2005 researchers have faced the problem of analysing trajectories according to space-time. The first studies on the analysis of trajectories offer an overview on how it is possible to analyze the trajectories starting from a set of POIs [72]; the analysis of trajectories from a set of points of interest (POIs) has been initially performed in [73, 72]. Over the years, these analyses have fed other different studies on trajectories, such as finding the probability of moving from one POI to another, using e.g. the Markov chains [7, 74], and then creating methods that predict the next movements of users by analysing their POIs.

Another type of analysis is the search for the similarity between the trajectories. In the literature we find this field studied in various ways according to the concept of similitude. In [72], two similar trajectories are defined as paths crossing in the same area several times, regardless of their time-spatial details, i.e. time-frame or direction. In [73], the authors identify two similar trajectories regardless of their geographical location (in fact, analyzing the problem that faces issues related to the analysis of videos or images).

Yet, high-density points have been studied very carefully [75], to carry out various analyzes on the dead spots (called standstill), in order to find e.g. the waiting time in places such as supermarket, bus stop or points of interest (like a museum). An improvement on the comparison between trajectories is the approach proposed in [25] that describes an algorithm to align the points of the trajectory with a map making the trajectory more homogeneous and precise.

Recent research, due to the introduction of GPS in the eigenvector and the need to tackle environmental pollution, analyzes the trajectories of the means of transport, such as taxis, facing the problem of traffic jams [76, 77, 78, 77, 76, 79]. For example, an innovation method is discuss in [78] which discusses how to exchange traffic information via a vehicle-to-vehicle communication connection.

However, none of the said studies aims at providing the quantitative amounts (such as e.g. number of people flowing in a trajectory, their average speed, etc.) that our approach achieves, which are useful for analysing, improving and proposing smart transportation services as well as logistic support.

POIs and recommendation systems: A recommendation system is a content filtering software that creates customized recommendations specific to the user so as to help him in his choices. Recommendation systems are divided into three main categories [80]: collaborative filtering, (CF), content-based filtering and (CB) and hybrid filtering (HF). Content-based filtering [81] makes recommendations based on user choices made in the past (for example, if a user likes to drink a carbonated drink like coca cola, the system offers recommendations for similar gaseous drinks such as sprite). Collaborative

filtering [82] allows users to give ratings about a set of elements, so that when enough information is stored on the system, it is possible to make recommendations to each user based on information provided by those users that have the most in common with them (for example, if Bob and Alan have seen the same horror films, one of the films of the same category seen by Bob is suggested to Alan and vice-versa). The hybrid technique is a mixture of the first two.

In the literature, several studies use collaborative filtering to suggest itineraries or Points of Interest (POIs). A POI is a well-known concept in literature [83, 84, 72], that is defined as an object associated with a latitude and a longitude which at least one person would reasonably be expected to have an interest or an utility. POI recommendation is one of the services available, suggesting places for users to visit [85]. In [86], the authors propose time-sensitive trip routes, consisting of a sequence of locations with associated timestamps. In [87] the authors propose a recommendation for itineraries based on multiple user-generated GPS trajectories. In [88] the authors propose a user-based collaborative filtering with time preference to explore user preferences on places visited and offer a recommended itinerary.

The weaknesses of such approaches are: 1) the works do not identify the point closest to the user; 2) recommendation systems are not updated in real-time; 3) the works do not automatically calculate points of interest. In contrast, our proposed work overcome such weaknesses.

2.3.3 Examples of Datasets

There are many datasets which are available for research purposes; these datasets help researchers to find new knowledge. Below is a list of datasets, they have been used to test approaches described in the drafting of the thesis:

- **Device Analyzer**: in 2013, Device Analyzer app was created [89, 15] and, thanks to the support of around 12,500 users, managed to collect a large amount of data on mobile devices from around 167 countries. Device Analyzer (DA) project and app aimed at collecting a large scale data of phone usage. It gathered data about running processes, wireless connectivity, the phones location, GSM communication, battery state and a number of system parameters. Users gave their contribution by downloading and installing an app that collects user data. Hence, a dataset that captures the real-world uses of Android devices was formed. All data were locally stored in a timestamped key-value store for periodic upload to a central server. DA app attempted to minimise its impact by scheduling uploads when the device was charging and with a 802.11 connection available. DA app captured more than 200

different events, the keys of data collected include: net (contains information about the state of network interfaces including the received byte number and sent byte number); location (position of device); power (information about the current battery); sms (contains information about incoming and outgoing text message); WiFi (contains information about currently visible WiFi networks). The designers of the DA offered us a set of data of over 950 users while respecting the privacy of contributors.

- **Geolife Trajectories**: this GPS trajectory dataset was collected in (Microsoft Research Asia) Geolife project by 182 users in a period of over three years (from April 2007 to August 2012). A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude [90, 24, 91]. The data are organized in folders; each folder identifies a user and for each user several files, each for the recordings on a day, are given. Each file contain several records, each with the following fields: Latitude; Longitude; All set to 0 for these datasets; Altitude; Date in number of days (after 12/30/1999); Date as a string; Timestamp. GeoLife has already been used several times in the literature due to data accuracy (about a value every 3 seconds) and the large amount of data [92, 90, 24, 91, 93].

- **T-Drive trajectory**: This dataset contains the GPS trajectories of 10,357 taxis during the period of Feb. 2 to Feb. 8, 2008 within Beijing. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches to 9 million kilometers [94, 95]. Each line of a file has the following fields, separated by comma: taxi id, date time, longitude, latitude.

- **Taxi Rome**: Dataset of mobility traces of taxi cabs in Rome, Italy. It contains GPS coordinates of approximately 320 taxis collected over 30 days [96]. Each taxi driver has a tablet that periodically (7s) retrieves the GPS position and sends it to a central server. Tablets have different version of Android and different brand. Despite its small size, the dataset has been used in several searches [97, 98, 99].

- **Taxi Trajectories**: Taxi trajectory data [100] is the GPS trajectories collected by 101 taxies equipped with GPS sensors in Beijing area [115.421387, 39.437614] x [117.321785, 40.609333] of longitude and latitude during a month, from 30 October 2010 to 30 November 2010. The positions saved by the mobile GPS device were almost one point per minute.

- **Truck Trajectories**: Truck trajectory data, referred to as Truck, is the GPS trajectories collected by 100 trucks equipped with GPS sensors in China during a period from August 2015 to October 2015. The space covered by the registered path is the large area $[86.882817, 0.230753] \times [172.467424, 43.405276]$. Each taxi is saved as a separate text file and each data point is saved as a line with: id; timestamp; latitude; longitude; speed; direction.

- **Yelp**: Yelp is a review forum that allows users to post their comments on a public place, restaurant or other services and to associate a number of stars ranging from 1 to 5 to give their own rating, in which the lower the number of stars the lower user's liking. The number of stars was used to evaluate the stability of the algorithm [101].

- **WiFi New York**: a list of New York City Wi-Fi hotspot locations. The dataset contains 3.345 elements, for each associated with a location in New York. The dataset was created on July 14, 2015 and the last update was done in April 4, 2019. Each record is valued by 30 columns. Among them we find: Type (for example free or limited free, Partner Site); Name (name of the WiFi); Location (area, for example: Bowne Park or Joyce Kilmer Park); SSID (the alphanumeric key that uniquely identifies the network); Latitude and Longitude (GPS coordinates).

2.4 Tools of protection

In response to the rapid spread of smartphone, attacks and user risks increase, therefore with the advent of the great Android giant and above all due to the large distribution of smartphones, in recent years the research has turned to the defend of the user, creating tools to protect the privacy and confidentiality of data. This implies a real need for tools that can detect malicious applications that steal information from users and this becomes increasingly complex as even benign applications read the private data and consequently it is not easy to estimate a priori if a data flow is legitimate.

In the next section we analyze the studies in the literature regarding user protection, in particular we discuss techniques to detect threats such as malware or data loss.

2.4.1 Related Work

Numerous researches have discussed how to identify privacy risks in mobile apps and designed privacy-enhancing technologies to mitigate the identified threats. Several studies have analysed permissions of apps, defining the authorisation as a serious security problem, mainly due to the misunderstanding of them by users [22, 13] who ignore potential

problems by choosing to trust an app store. Specifically, in [13], it is shown how an ignorant user or an organisation can fall prey to an app. Android OS adopts the "principle of least privileges", however apps are often overprivileged [20, 40]. As a consequence, the permission system can hardly guarantee realistic security [22]. According to [23], it seems that there are no visible differences in the permission list used in malware with respect to the ones used in benign apps. The absence of diversity makes classification more complex and highlights that checking how permissions are used at runtime can be more effective.

Thus, an Android app can seriously threatens security and the user in terms of escalation of privileges, remote control, price theft and loss of privacy. An app that has access to corporate data is a potential channel for such threats.

Techniques for detecting malware can be divided into three main categories [102]:

1. static analysis methods: check the code without running it;
2. dynamic analysis techniques: monitor execution and inspect interactions with the outside, thus detection is performed by collecting data at runtime [103].
3. hybrid analysis techniques: combine both static and dynamic analysis methods.

Dynamic analysis of malware has received a lot of attention in the research community. Analysis systems such as CWSandbox [104], and Anubis [105] execute malware samples within an instrumented environment and monitor their behaviour for analysis and development of defense mechanisms. Most dynamic analysis techniques mainly operate when samples are unpacked. Furthermore, static and dynamic analysis solutions are mainly based on:

- signature, i.e. the identification of unique signatures that define the malware,
- heuristic, i.e. rules determined e.g. by machine learning techniques or by experts are used to detect malware.

Recently, classification algorithms have been employed to automate and extend the idea of heuristic based methods. In these methods, the binary code of a file or the behaviour of an app during execution is represented and some classifiers learn corresponding models in order to classify new (unknown) apps as harmful or benign [106, 107, 108]. There are already several frameworks that identify a possible attack on the device including Andromaly [27], a framework that realizes a host based malware detection system that continuously monitors various features and events obtained from the mobile device and then applies some machine learning based detectors to classify collected data as normal

(benign) or abnormal (harmful) [27]. Moreover, in [109], it has been determined whether a newly requested permission type is benign or not by using machine learning techniques based on previous user preferences.

Many methods are based on permission monitoring, e.g. among static methods, there are: Adrienne [35], a tool to detect over privilege in compiled Android apps; AndroidLeaks [12], automatically detecting potential privacy leaks in apps based on maps of permissions and APIs; Drebin [110] implementing a classifier based on APIs and other apps data; Fan [111], proposing malware detection systems based on API log data mining. Yet, FlowDroid [41] is a tool that exercised static data flow analysis to find dangerous code in Android apps. Other research works have looked for malware according to the past behavior of the user [112, 109].

However, the available tools implementing their proposed techniques have three significant flaws: 1) they are not very user-friendly as they must be downloaded separately and subsequently it is only possible to test the application through the command line. For this reason they are not very efficient for average users. 2) The common ground of these methods is to find a correct measurement to identify malware; all these works are based on already known behavior or signatures. This is a limitation to identify new threats not yet known or little used. 3) these tools are based on the analysis of the study of harmful permissions, as they are more relevant than the other Android levels "in particular normal-level", but this is a limitation as there are, to date, attacks that can be based on normal permissions, as we will see in the following chapters.

2.5 Conclusion

In this chapter all the bases and related works have been presented, to understand and evaluate the contributions presented in other chapters.

The great success of the Android devices has been confirmed in literature but at the same time the big gaps in terms of security has been widely documented. Attacks on smartphones and user privacy risks have been introduced. These attacks and risks arise mainly from the continuous connection to the network and the gaps in the Android permissions system.

In addition, techniques for the creation of services for citizens have been illustrated based on user profiling. These services also come to life thanks to the various information available from the devices via API. To reinforce this notion, several datasets have been listed that will be used in the experiments present in this thesis work to create attacks, services and user protection tools.

Chapter 3

Manipulation of user data via network

Smartphones are the center of attention of millions of people; the mobile device is used for multiple reasons (address book, e-mail clients and many more) and offers services of several type and genre, satisfying the tastes of the whole community. Generally, these services require an active connection to keep up to date and to share information on the network. This constant and continuous connection to the network creates various problems for the user's security due to the leakage of their information which occurs through the use of API.

The huge use of smartphones allows new services to be created every day to facilitate the user in their daily life by offering applications to calculate the traffic density, calculate the number of people in a building, discover points of interest, etc. Many of these services have a knowledge base take from devices such as tablets, smartphones or TVs. For this reason, data from smartphones is widely sought after and used (see Chapter 2.3.3). Simultaneously, via API, it is possible to extract sensitive user information and forward it on the network. This mechanism, protected in part by the permission system, is an attack point that exploits various permission weaknesses to access user information (see Chapter 2.2.1).

In this chapter we focus on the potential of analyzing data taken from devices. We present two services for citizens that are based on the use of GPS data taken from smartphones and a possible attack on smartphone users highlighting the feasibility of the approach:

- the first service proposes an approach that identifies the flows of people from collected GPS data. This in turn enables us to compute significant parameters, such as people average speed, amount of travelling people, etc. A proper solution for data filtering and analysis has been implemented and tested against real data, which reduces as much as possible running time by lowering the number of needed comparisons. The ability to gain insights on people flows can have many outcomes in the area of smart transportation, e.g. the efficacy of transportation means can

be assessed, then potential improvements can be suggested on public transportation means, infrastructures, etc. In this study, our contributions include: provide a method capable of extrapolating a large amount of data providing the quantitative amounts which are useful for analysing, improving and proposing smart transportation services as well as logistic support or to deal with traffic jams. This makes the algorithm multi-functional, as we will see during the test phase.

- the second service illustrates a method for finding a set of Points Of Interest (POIs), which we determine using the DBSCAN algorithm. In this study, our contributions include a solution to identity POIs through a set of trajectories. The method is the search for Points Of Interest from user trajectories and not from a predefined list.

- the approach describes an attack able to track the user position by known WiFi (open and private); based on permission levels and mechanisms proper of Android OS, this section proposes an approach whereby an app attempting to connect to WiFi networks could reveal the presence of some known access points, thus the geographical location of the user, while she is unaware of such a feature. This is achieved without resorting to GPS readings, hence without needing dangerous level permissions. This work presents a new attack to Android mobile devices that can determine geographical location by attempting WiFi connections, which users are unaware of. Unlike previous studies, the attack comes from a misleading app, installed on the device, that forces WiFi to be enabled. Our approach is compatible with all Android OS versions, despite the security policies of the latest version, and does not require dangerous-level permissions.

All approaches are based on *GPS coordinates*. In the first two case, the coordinates are offered by the user, while in the third approach the coordinates are identified in an illicit but feasible way, highlighting the vulnerabilities of Android devices.

3.1 Identification Service of Spatio-Temporal Flows

The wide-spread use of mobile devices provides us with useful data on people GPS positions. This section aims at proposing a solution for analysing the trajectories of users and gain knowledge that can be used to enhance public transportation, transportation planning, both communication and transport infrastructures, etc.

An approach is proposed for revealing significant quantitative characteristics of people movements, such as e.g. the largest amounts of people travelling between several parts of the city, their average speed, and the very frequently used paths. For the said aim, we

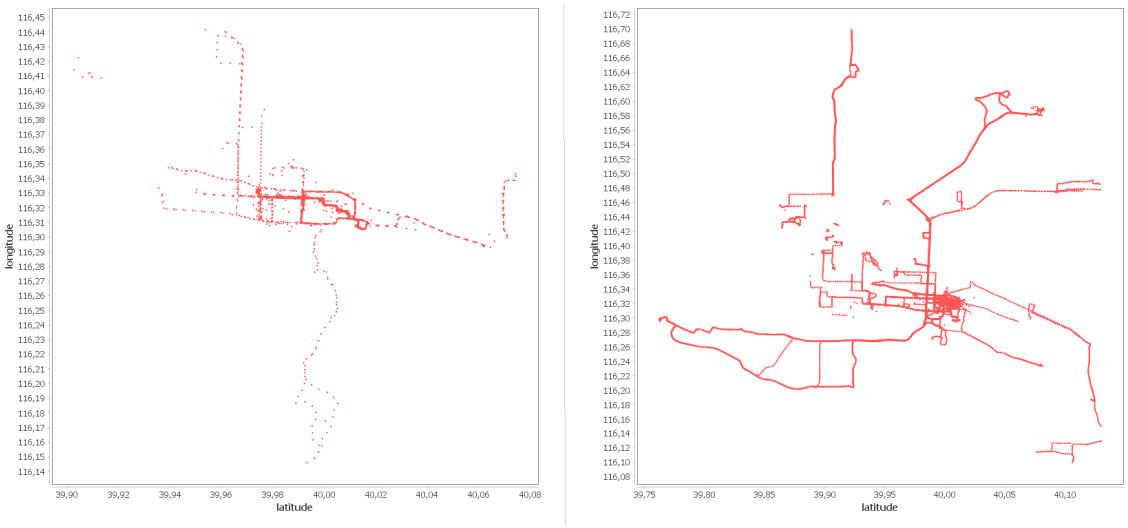

Figure 3.1: Two examples of trajectory according to the first definition. Each trajectory is referred to one user.

devise novel automatic means to "clean" and analyse data; starting from row data representing GPS positions, cleaning aims at safeguarding us from errors in GPS measures and repeated points. Then, data analysis determines spatially similar paths for different users and computes their length in kilometers. Furthermore, the algorithm automatically identifies the number of users in frequent paths. Such a quantitative analysis offers means to determine long and shared paths.

Data gathered by this solution could be used to improve services for the citizens in several ways. E.g. during an evacuation in case of emergency it would be possible to determine, and then check, clear and assist the most used paths. The experiments section illustrates some application fields. For a better understanding of the approach, we give a set of definitions like the definition of trajectories or similarity.

According to [113], a trajectory is formally defined as follows:

Definition 1. A GPS trajectory is a list of GPS data points $\{p_0 = (x_0, y_0, t_0), p_1 = (x_1, y_1, t_1), ..., p_n = (x_n, y_n, t_n)\}$, where $\forall i \in [0, n],\ p_i = (x_i, y_i, t_i)$ with $t_i < t_{i+1}$, and x_i, y_i and t_i represent longitude, latitude, and timestamp, respectively.

Spatial similarity is defined as follows.

Definition 2. Two trajectories are spatially similar if they pass through the same n points in the same temporal order. In other words, given two trajectories $T_1 = \{a_0, a_1, a_2, \ldots, a_n\}$, $T_2 = \{b_0, b_1, b_2, \ldots, b_n\}$ they are spatially similar if it exists a set of points $s = \{p_0, p_1, p_2, \ldots, p_n\}$ for which $s \in T_1$ and $s \in T_2$.

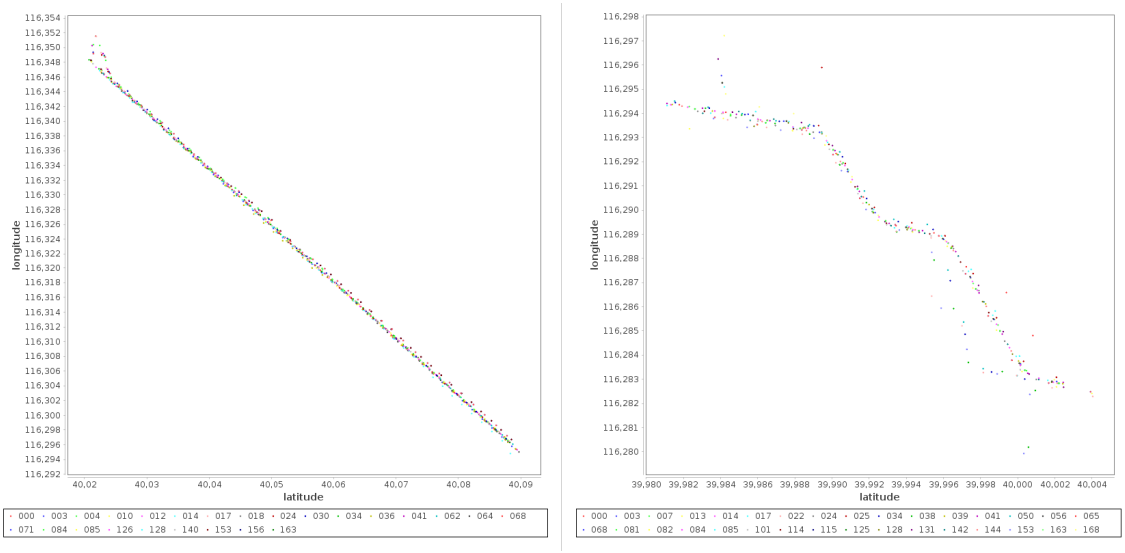

Figure 3.2: Two examples of flows. In each example, each color is referred to one user. In first example, the flow density is equal to 25; in the second example, flow is equal to 32

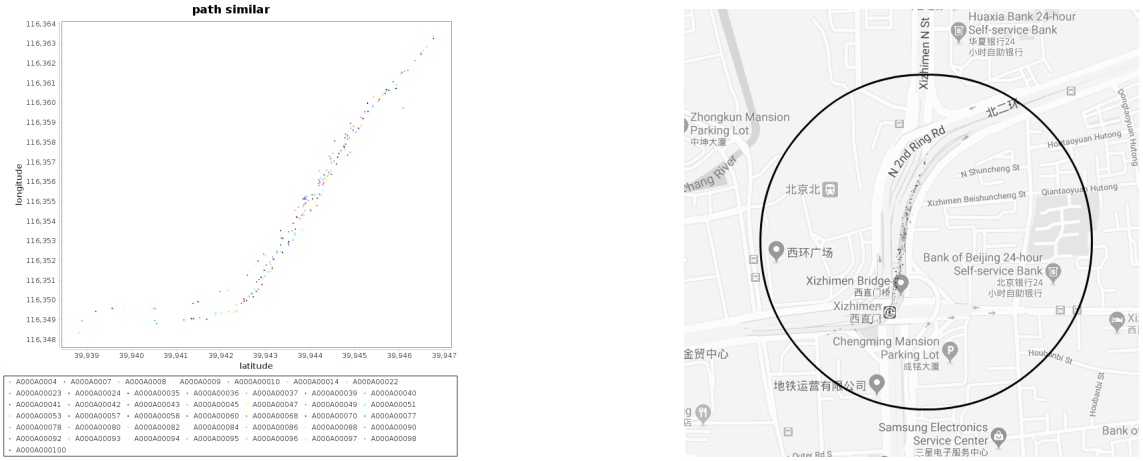

Figure 3.3: Integration of a flow in a map. The density is equals to 43 and the distance is equals to 1.1 kilometers.

According to [113], the *distance* $d(p_i, p_j)$ between point p_i and point p_j, when their coordinates are given by latitude (lt) and longitude (lg), is defined as follows.

$$d(p_i, p_j) =$$

$$2R \arcsin \sqrt{\sin^2 \frac{lt_i - lt_j}{2} + \cos lt_i \cos lt_j \sin^2 \frac{lg_i - lg_j}{2}} \tag{3.1}$$

where R is a constant, given by the Earth radius. This formula distance called HAVERSINE FORMULA, yields high precision due to the use of constant R.

For all the definitions comparing coordinates, we consider that two points are overlapping (equal in a mathematical sense) if their distance computed using the above distance formula is less than $300m$. This distance is considered the upper limit distance for two consecutive bus stops, hence let us find the amount of people transiting in an area having a radius of $150m$, which is useful to determine whether to serve them with the same bus stop[1].

We further define *flow* and *flow density*.

Definition 3. A flow is a continuous and uniform movement of people in one direction on the trajectory.

Definition 4. A flow density is the number of users travelling through the same flow.

For our purposes we have identified, at a 4-hour interval from each other, the following six time slots:

1. from 00:00 to 03:59;
2. from 04:00 to 07:59;
3. from 08:00 to 11:59;
4. from 12:00 to 15:59;
5. from 16:00 to 19:59;
6. from 20:00 to 23:59.

The proposed algorithm is independent of the above time slots, which can be freely set according to the analysis at hand.

We define the *temporal similarity* as follows.

[1]This parameter as well as others that can be found in the remainder of the section can be configured and tuned for the analysis at hand without loss of generalisation for our approach.

```
public void clearData() {
        order_trajectory();
        deleteOutline();
        clearTrajectory();
}
```

Figure 3.4: Data cleaning code which includes trajectory sorting, removal of outliers and noise removal.

```
public void order_trajectory() {
    Collections.sort(this.getTrajectory());
    long tmp = 0;
    for (Point point : this.getTrajectory()) {
        if(tmp < Long.parseLong(point.getData())) {
            tmp = Long.parseLong(point.getData());
        }
    }
}
```

Figure 3.5: Code sorting the trajectories based on the timestamp.

Definition 5. Trajectories that are spatially similar are said temporally similar if the people movements occur in the same time slot.

It is possible to see two examples of trajectories in Figure 3.1, two examples of flows in Figure 3.2 and an example of integration of a flow in a real map in Figure 3.3.

Section 3.1.1 explains the described approach, taking advantage of the two steps for the pre-processing phase. Subsequently the Section 3.1.2 illustrates experiments and results with the possible scenarios of action.

3.1.1 Algorithm to determine Flows of People

The proposed data analysis algorithm aims at achieving the set of similar trajectories that are the most shared among users and higher than a given threshold. To do this, two phases were carried out: **data cleaning phase** performs a data cleanup (see Section 2.3.1) and **calculation of trajectories** identifies the Spatial Temporal flows.

1) DATA CLEANING PHASE: the significant fields retained in our study are: latitude, longitude and timestamp. For a given trace, data points have been sorted according to their timestamp, in order to find a trajectory out of recorded points (see Definition 1). For each user, its recorded data have been considered as a whole. Hence, for a user all the positions and movements are available as one trajectory. Each trajectory has been "cleaned

```
public void clearTrajectory() {
    int distanceMin = 150;
    ArrayList<Point> tmp = new ArrayList<Point>();
    for (int i = 0; i < points.size() - 1; i++) {
        int indice = i;
        tmp.add(points.get(i));
        while (i < this.points.size() - 1 &&
            (points.get(indice).calculate_distance_in_meters(points.get(i + 1))
            < distanceMin)) {
            i++;
        }
    }
    points = tmp;
}
```

Figure 3.6: Code and example of removal of noise. In the code (at the top) you can see the code written in Java 8 in which points that are less than the value set in DISTANCEMIN are removed. Below, in the figure, an original trajectory with high density points (left) and its transformation in a trajectory with uniformly geographically-spaced points.

up" by removing outliers and noise. Since GPS measures can be affected by some error, e.g. in urban centers due to skyscrapers [114], a search for anomalous shifts, hence outliers, was carried out on each trajectory. Each point was compared to the n previous points and n subsequent points (we have set n as 5 in our experiments) to detect abrupt changes in speed, and if this is the case, the anomalous point was considered an outlier and removed. Noise removal aims at finding points that are too close to each others, in order to offer a more homogeneous trajectory having almost equally geographically spaced points (instead of equally time-spaced recordings). Therefore, the points that are less than 150 meters between each other have been removed (distance is computed according to the Formula 3.1).

Figure 3.7 shows on the left an initial trajectory with high density points, and then on the right a trajectory resulting after executing the second cleaning step, i.e. the removal of high density points. Overall, data pre-processing and cleaning has a fundamental role because it offers us simpler data to manage and cleaner data.

Figure 3.6 shows that given an initial trajectory (on the left), the first cleaning step removes points with sudden changes in speed from the trajectory comparing it with the 5 previous points and the 5 further points. Once the anomalous point has been identified, it will be removed from the trajectory (on the right). Below each image we find the list of

```
1  public boolean deletePoint(Point p) {
2      for (int i = 0; i < points.size() - 2; i++) {
3          String vehicle_A = points.get(i).locate_vehicle(points.get(i +
               1));
4          String vehicle_B = points.get(i +
               1).locate_vehicle(points.get(i + 2));
5          if (!vehicle_A.equalsIgnoreCase(vehicle_B)){
6              if (checkNear(i + 2)) {
7                  points.remove(points.get(i + 2));
8              }
9          }
10     }
11 }
```

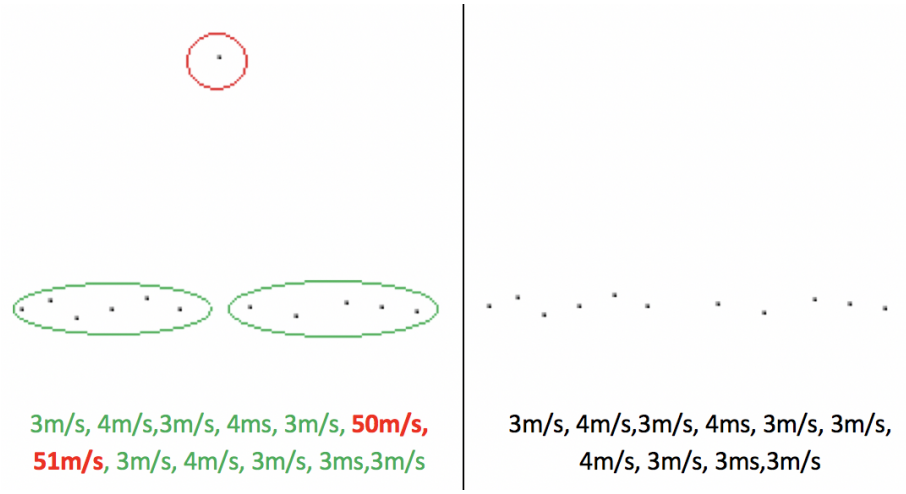

Figure 3.7: Code and example of removal of outliers. In the code (at the top) you can see the code written in Java 8 in which line 3 and 4 hypothesizes the vehicle that is being used to check the speed change; if there is a change of vehicle for a single moment the point is considered an outlier. In Figure (in the bottom) you can see an example of anomalous points highlighted in red that is removed from this step.

speeds from one point to another from left to right.

In the Figure 3.4, 3.5, 3.7, 3.7 we show the code snippet for the Data Cleaning Phase.

2) CALCULATION OF TRAJECTORIES: The calculation of the trajectories involves the calculation of the following parameters: (i) the amount of people travelling in the same trajectory, i.e. the flow density; (ii) the set of spatio-temporal similar trajectories that are the most shared by a minimum number of users; and (iii) the average speed on each trajectory.

For searching spatially similar trajectories, users are analysed in pairs to identify whether they share GPS points. This computation may require a considerable amount of time, as execution time is proportional to the number of users to be analysed and the length of their trajectories. Consequently, an advanced algorithm has been devised that

```
1 List<Integer> pts =
2    IntStream.range(0, t1.getTrjSize())
3       .boxed()
4       .filter(i -> t2.getTrj().contains(t1.getTrj().get(i)))
5       .collect(Collectors.toList());
```

Figure 3.8: The algorithm written in Java8 to search for similar trajectories, where t1 and t2 are two trajectories.

reduces the number of comparisons and can be run in parallel. To ease parallelism and avoid conflicts, our algorithm uses Java 8 stream processing libraries, hence it scales to any number of cores and threads, providing a higher speed-up when a higher number of cores becomes available [115].

Firstly, the algorithm (see Figure 3.8) gathers the indexes of points, as list pts, occurring in both trajectories, lists t1 and t2, which are within $200m$. Specifically, line 2 enumerates i from 0 to the number of points (i.e. length) of t1, and line 4 keeps the index i of the point found in both t1 and t2. Secondly, from all indexes in pts, it checks whether the shared points found form a path, by comparing their mutual distance with $200m$. Finally, it takes all the longest paths above a threshold of $1km$.

Checking whether any point is in both trajectories of length n requires performing comparisons on the order of n^2. Whereas, finding subsequences of points having length $n/2$ requires performing comparisons on the order of n^3. Therefore, the first solution is faster than the second.

Once a shared trajectory has been found, we identify how many users pass for that trajectory using again the code above. Then, parameter t1 represents the trajectory found and t2 represents the trajectory of a generic user. When the resulting list has the same number of values as t1 then the user corresponding to t2 is counted as passing in trajectory t1. Now, if the spatially-similar trajectory has a density greater than 10% of the total number of users, it is said a flow. Moreover, for each trajectory labelled as a flow, we count the number of users passing on the trajectory on each defined time slot, and also compute the average speed of users. For each trajectory, the time-stamp of its first point lets us find the time slot it belongs to. The speed is computed by the ratio between travelled meters and needed time (difference between the last and the first time-stamp of the trajectory).

3.1.2 Experiments and Results

The approach has been tested against several datasets (see Table 3.1 and Chapter 2.3.3 for details):

Table 3.1: List of datasets used for experiments.

Dataset	GPS points	Users	Period	The sampling rate	Area
Taxi Trajectories Data	4,297,047	100	from Nov 1, 2010 to Nov 30, 2010	one point per minute	Beijing
Truck Trajectories Data	10,059,584	100	from Aug. 2015 to Oct. 2015	one point per minute	China
Geolife GPS Trajectories	18,021,911	182	from Apr. 2007 to Oct. 2011.	each 1-5 seconds or each 5-10 meters per point	Beijing

1) GEOLIFE DATASET: Data gathered by GPS devices and recorded have to be transformed before applying the actual data analysis in our approach. Transformation is meant to remove useless pieces of data, and anomalous data. This helps speed up the following data analysis. The steps to be performed on data and constituting preprocessing are as follows.

1. Conversion to CSV: the GeoLife project offers the data in a PLT format, therefore the first step is to convert them to CSV to have more control on data (i.e. it is easier to visualize data and perform checks).

2. Removing superfluous fields: the fields 3, 4, 5, 6 (described in the previous Section 2.3.3) have been removed, as useless for our proposed analysis and our purposes.

3. Sorting positions and achieving trajectories: the available recorded positions of a given trace have been sorted according to the timestamp field, in order to achieve a trajectory out of the positions.

4. For each user, all of its recorded data, which were available as several files, one for each day, have been considered as a whole. Hence, for a user all the positions and movements are available as one trajectory (for that user).

Our pre-processing and cleaning operations ("Data Cleaning Phase") managed to remove around 400 outliers for each trajectory; and between 60% to 90 % of points, due to waiting areas, for each trajectory. The algorithm, illustrated in Section 3.1.1, takes in input 182 trajectories and as a reference trajectory a user (which can be chosen by taking

Table 3.2: Results of the data analysis with the Geolife dataset.

meters	people	slot 1		slot 2		slot 3		slot 4		slot 5		slot 6	
		p	s	p	s	p	s	p	s	p	s	p	s
2650	22	17	2.81	13	1.08	16	1.16	14	3.01	6	0.22	11	0.07
3052	19	9	4.80	8	2.22	11	2.58	1	5.43				
2046	39	25	0.05	28	0.43	15	2.45	16	4.21	1	0.03	6	6.69
2645	44	24	6.45	24	13.18	21	11.35	10	5.24	3	11.27	12	12.74
2407	33	10	14.97	10	11.08	10	11.51	2	16.42	1	21.46	5	9.54
2566	19	10	4.12	10	12.97	10	8.72	2	16.89	3	11.12	3	0.52
3123	18	6	3.53	6	10.14	9	7.66	6	8.26	2	11.61	2	0.23
2037	19	11	12.07	10	8.63	3	8.12	5	7.30			1	10.64
2092	18	23	10.98	26	6.86	24	16.62	11	19.33	7	8.97	10	3.35
7431	18	7	16.89	11	24.93	9	17.16	6	23.06	3	21.48	3	21.85

the user with the longest trajectory) and returns all flows having density is greater than 10 % of the number of users.

Table 3.2 shows a sample of the results found by our data analysis. Each row corresponds to each single flow found, and gives: the distance travelled (the distance between the starting point and the end point of the flow); the flow density, i.e. the number of different users travelling on that flow (in column denoted by people); and for each time-slot (from 1 to 6), the flow density (denoted by p), and the average speed in m/s (denoted by s). The total flow density counts how many different people have travelled in the whole day, hence it does not count how many times in different time slots, which is instead in the flow densities for each time slot. Some flows have the length of a few kilometers and a low average speed, this could correspond to areas having a high traffic or that are crossed on foot.

Figure 3.9 shows a trajectory that has a very low average speed (0.21m/s). The density of the flow is equal to 25. The area is almost 2.3km in lenght. This finding suggests that the adoption of a public transportation means would assist users previously travelling on foot or lessen traffic congestion.

Figure 3.10 shows two flows found by our analysis. The first flow covers the space between the two bus marks on the bottom-part of the figure and is about 2km. The second flow is given by all plotted points and covers about 2.7km. This finding could be used to propose one bus line for the combined path, and a bus stop where we have found that one path stops.

So, results shown that it is possible to identify relevant flows. We detected 25 flows involving more than 10% of the users. In addition we have also found some paths involving more than 30% of the users.

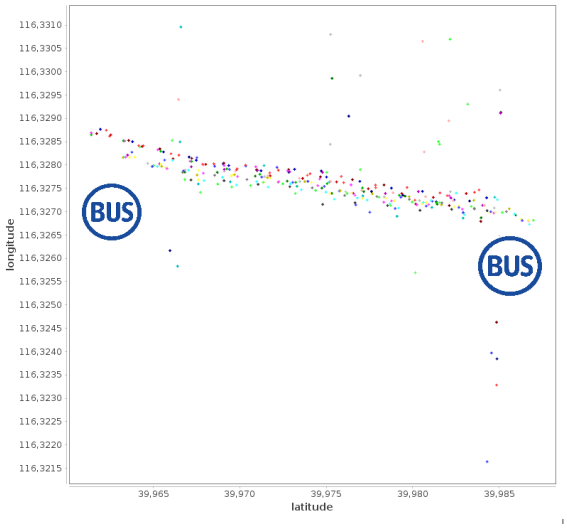

Figure 3.9: An area where traffic goes at low speed. Each color is an user.

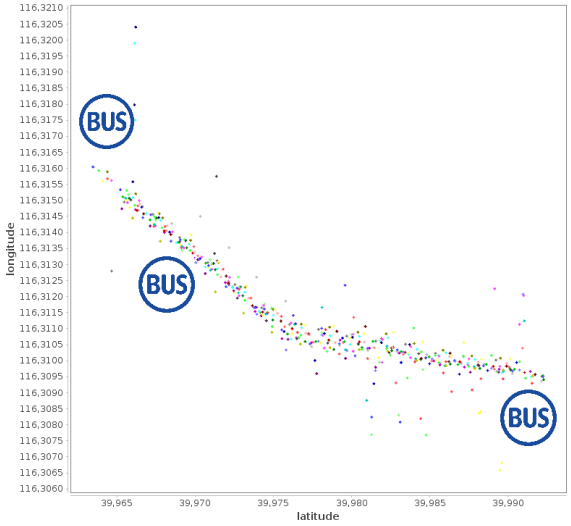

Figure 3.10: An area where traffic can be improved by combining trajectories and proposing a public transport means.

2) TAXI DATASET: Taxi trajectory data [100] is the GPS trajectories collected by 100 taxies equipped with GPS sensors in Beijing area [115.421387, 39.437614] x [117.321785, 40.609333] of longitude and latitude during a month, from 30 October 2010 to 30 November 2010. The positions saved by the mobile GPS device were almost one point per minute.

Each taxi is saved as a separate text file and in these files, a data point is saved as a line with the format: id; timestamp; latitude; longitude; speed; direction. For tests, the last two fields were excluded and each trajectory was reordered based on the timestamp as the previous dataset.

The algorithm in Section 3.1.1, takes in input 100 trajectories of taxi. The first step was data pre-processing. Thanks to this step, it lightened the data of around 400 outliers and eliminated about 70% - 80% by removing the noise (for each trajectory).

Once the data processing is complete, the algorithm starts with the identification flows. For a comparison with the first methodology, the temporal flows were initially ignored. Therefore, it has obtained about 80 flows with a minimum distance of 1 km and with a minimum number of users equal to 8% (or 8 out of 100). Subsequently, an estimate was made of the time required to direct agents towards the time slots suggested to users. In this case it has been verified that the preferred time slot is between 08:00 and 11:59, but each POI can be associated with its own time slot.

Finally, in general, the algorithm takes a trajectory as a reference to the extrapolation of flows, in this case the tests were performed taking about 10% of the trajectories (randomly chosen) and all the results were assembled together. In the Table 3.3 the details of the experiments. Examples of flows are shown in Figure 3.11, Figure 3.12 and Figure 3.13.

3) TRUCK DATASET: Truck trajectory data, referred to as Truck, is the GPS trajectories collected by 100 trucks equipped with GPS sensors in China during a period from August 2015 to October 2015. The space covered by the registered path is the large area $[86.882817, 0.230753] \times [172.467424, 43.405276]$.

As the previous dataset, each taxi is saved as a separate text file and each data point is saved as a line with: id; timestamp; latitude; longitude; speed; direction, but for tests, the last two fields were excluded and each trajectory was reordered based on the timestamp.

In this case, the algorithm was tested with a set of Truck trajectories. The first step was data pre-processing. Thanks to this step, for each trajectory, it has lightened the data by approximately between 0.001% and 0007% by eliminating outliers and by around 70% - 80% removing the noise. In this case, a minimum distance between two points was set equal to 250 meters for elimination of noise; this choice was made due to the high number

Table 3.3: The following table presents an overview of the tests performed with three datasets. Each record contains: the name of dataset used (*dataset*); the reduction of the number of points in the trajectories thanks to the phase of preprocessing of the data containing the minimum distance between two points for the removal of noise (*Distance*), the percentage reduction of the points thanks to step 1 (*Noise*) and the percentage reduction of the points thanks to step 2 (*Outliers*); the time slot most chosen by users (*Time Slot*); the number of flows found (*Number of flows*); the maximum density detected among flows (*Maximum Density*); the maximum distance in meters of the longest flow found (*Maximum distance*). All the flows detected have a minimum number of 8% of the total number of users and a minimum distance of 1 kilometer.

Dataset	Preprocessing			Time Slot	Number of flows	Max Density	Max Distance
	Distance	Noise	Outliers				
Geolife	150	90%	0,004%	00:00-07:59	25	53	3052
Taxi	150	80%	0,002%	08:00-11:59	79	70	1716
Truck	250	60%	0,002%	16:00-19:59	114	31	1718

of points present in each trajectory and almost allows to halve the response times of the algorithm with a minimum number of derisory flows.

Subsequently the algorithm started by taking 10 trajectories at random and using them as a reference for a comparison between all other trajectories. Therefore, it has obtained about 114 flows with a minimum distance of 1 km and with a minimum number of users equal to 8% (or 8 out of 100). For these test, it has been verified that the preferred time slot is between 08:00 and 11:59, but, even in this case, each POI can be associated with its own time slot. In the Table 3.3 the details of the experiments. Examples of flows are shown in Figure 3.14, Figure 3.15 and Figure 3.16.

Table 3.3 reports the results. Each result can be associated with a specific area based on the data used:

- Geolife trajectories: it is possible to use these data to create suggestions for urban services in the city; that is starting from flows of people move on foot, by means of our algorithm, it is possible to identify flows with high density from one point to another in the city, suggesting bus stops (see Figure 3.9 and Figure 3.10).

- Taxi trajectories: it is possible to exploit the algorithm to prevent traffic jams, since the set of trajectory analyzed encloses a set of taxi routes. By reveling the areas where generally more taxis can be found (identified by a high density flow), it is possible to create statistics on road traffic.

- Truck trajectories: similarly as for taxis, it is possible to make forecasts on traffic but in this case a lower number of entities within the flow is sufficient since the

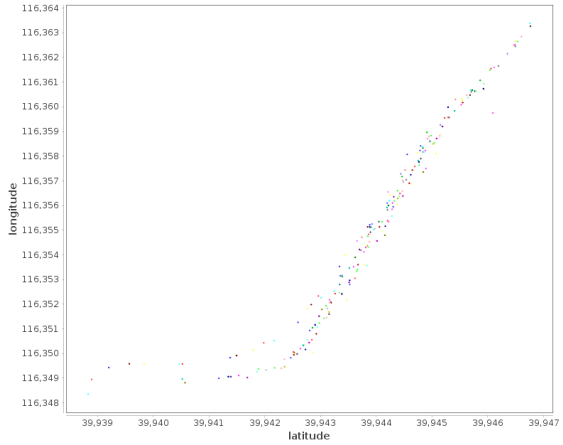

Figure 3.11: Flow with total length of 1075 meters and a total number of users of 43. Each color corresponds to a taxi ride.

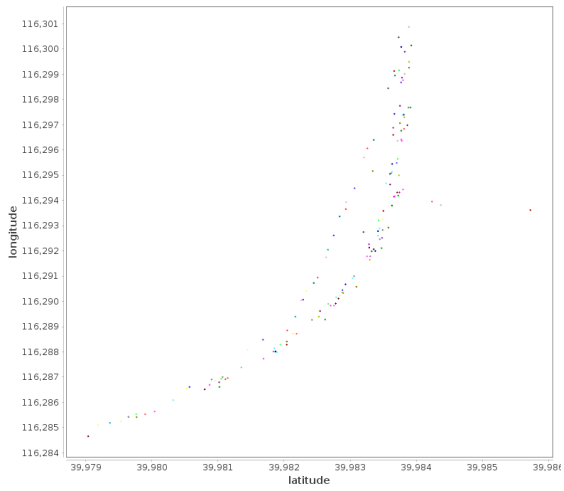

Figure 3.12: Flow with total length of 1158 meters and a total number of users of 28. Each color corresponds to a taxi ride.

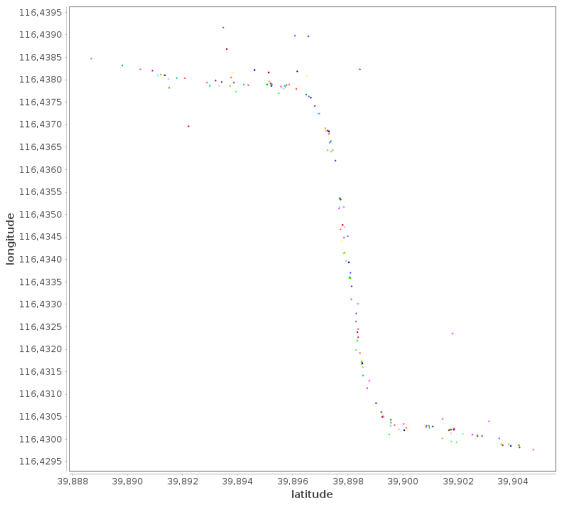

Figure 3.13: Flow with total length of 1570 meters and a total number of users of 20. Each color corresponds to a taxi ride.

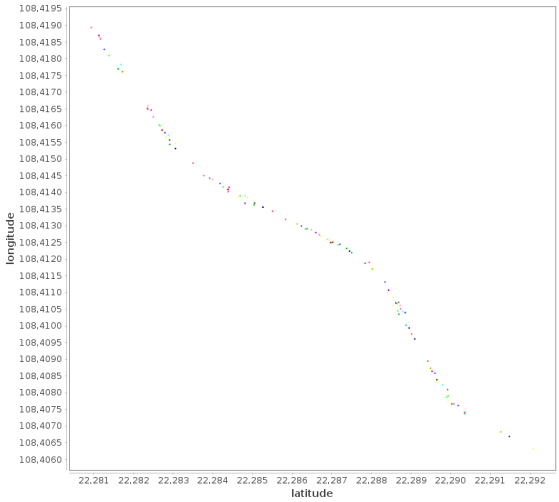

Figure 3.14: Flow with total length of 1208 meters and a total number of users of 19. Each color corresponds to a truck ride.

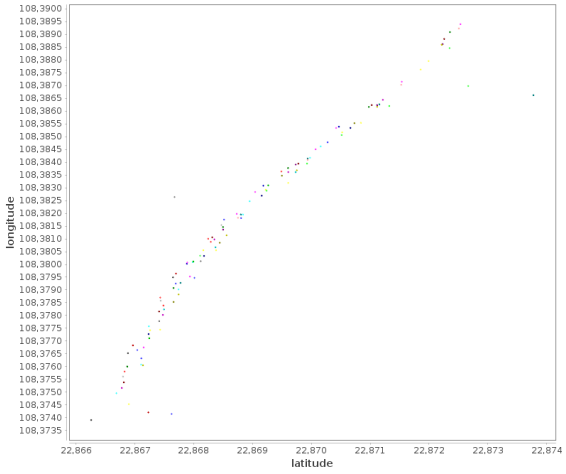

Figure 3.15: Flow with total length of 1255 meters and a total number of users of 28. Each color corresponds to a truck ride.

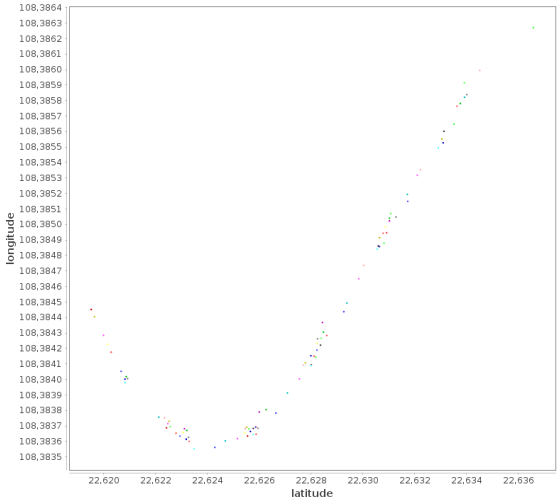

Figure 3.16: Flow with total length of 1372 meters and a total number of users of 18. Each color corresponds to a truck ride.

objects under consideration are trucks and they lead regardless of traffic slowdown due to their large dimension.

Finally, using this approach, it is possible find Points Of Interest in the city. Starting from these flows, it is possible to extrapolate the Points Of Interest by making a match between the POIs of the analysed city (removable from datasets present on the web) and the points close to each detected flow (with a maximum distance of 100 m).

3.2 Point of Interest Detection Service

Given the extraordinary use of mobile devices and various technologies tracing one's geographical position, it becomes increasingly easier to acquire information relating to users GPS in real time. This availability has triggered several studies based on user positioning, such as the analysis of the flows of people in the cities [6] or the prediction of people movements [7]. This has also led to the improvement of services that identify the Points Of Interest for a city to offer benefits to users who want to find a specific place.

A Point of Interest, commonly abbreviated POI, is defined as an object associated with a latitude and a longitude which at least one person would reasonably be expected to have an interest or an utility.

This section proposes an approach to identify POIs from an automatic analysis of a real dataset offered by the Geolife experiment [91] and others. The points found by our analysis were verified by matching the results with Google maps data. It was confirmed that they correspond to real POIs, i.e. parks and restaurants, hence validating our approach.

3.2.1 Methodology for Determining POIs

An extensive series of experiments has been performed in order to study the movements of users in different situations. In this section, a method to extract StayPoints (SPs) is described and then a clustering algorithm is used to find important places, called POIs, as it can be easily seen that places that are of interest attract several persons at the same time. The algorithm is divided into two steps: *Data Cleaning and Grouping of Trajectories* and *StayPoints detection*:

1) DATA CLEANING AND GROUPING OF TRAJECTORIES: the first step in our analysis was the data cleaning of the trajectories, based on the speed of their GPS points, with the goal to remove inconsistent data. Considering a trajectory, a sequence of GPS points ordered by the time of recording, we computed the velocity of a point as the ratio of the

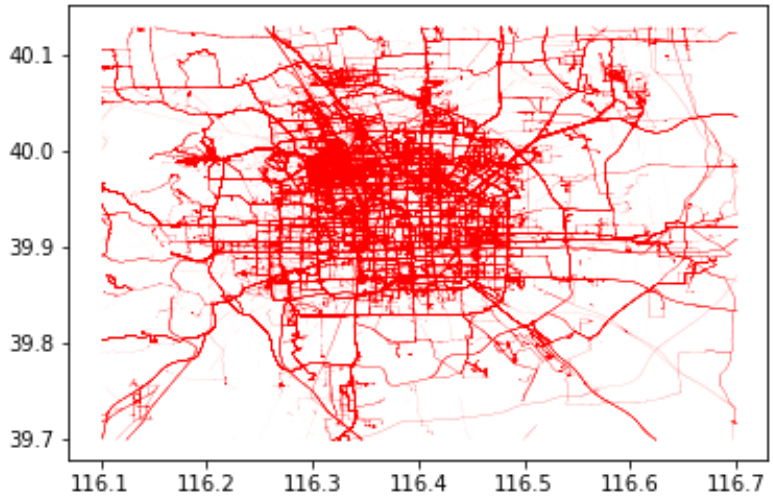

Figure 3.17: Plot of clean trajectories in the selected Beijing metropolitan area. Axis: X=longitude, Y=latitude.

distance from it and its consecutive (applying the Haversine formula illustrated in Equation 3.1) and the difference of time recording them. If this velocity exceeds 100 m/s, the second point was deleted. Another case is when the velocity appeared in the form 0/0, this noise was caused by the GPS device that did not run properly.

Furthermore, by plotting the trajectories, we can see a second problem: some paths appear broken (not continuous) probably due to the presence of buildings or tunnels that disturb the GPS signal (in some areas the recording is lost). The filtered data were then grouped into 6 time slots of 4 hours each: Slot1 [00:00:00, 03:59:59], Slot2 [04:00:00, 07:59:59] and so on, in order to analyse the movement during different time slots.

Related to experiments with the Geolife dataset, information about these slots are shown in Table 3.4 and for time Slot 3's trajectory data we have drawn the GPS data on the map to get a rough idea of the users' activity in this period of time in this area (see Figure 3.18).

2) STAYPOINTS (SPS) DETECTION: For every slot of time, after grouping trajectories by users, the second step of our work was the StayPoints (SPs) detection [116]. When we find a region in which a user has spent a considerable time on its surroundings, the centroid (the mean of coordinates of the points belong to it) of this cluster represents an SP. The algorithm that we implemented for the SP detection needs as input a TimeThreshold and a DistanceThreshold. In general, if an individual stays over 20 minutes (TimeThreshold) within a distance of 200 meters (DistanceThreshold), a SP is detected.

We obtained many SPs for every time slot, as shown in Table 3.5. For the Slot 3's trajectories the plot of their SPs is in Figure 3.19. Then, we focused on POIs that cluster

Table 3.4: Information about different time slots

Time Slot	**Total number of**		
	GPS Points	***Trajectories***	***Users***
1	3978234	5878	156
2	3729429	4302	150
3	4976744	6613	166
4	3107232	4702	168
5	889076	1505	114
6	1341196	2537	129

Data: A trajectory T={p_i,p_i+1,...}, a distance threshold (DistThr) and time span threshold (TimeThr)
Result: A set of stay points SP ={}
i=0, cardinality_traj= $|T|$
while $i < cardinality\_traj$ **do**
    $j = i+1$;
    while $j < cardinality\_traj$ **do**
        dist=distance (p_i, p_j);
        if $dist > DistThr$ **then**
            δtime=time_p_j−time_p_i;
            if $\delta time > TimeThr$ **then**
                S.coords=MeanCoords($\{p\_k | i < k < j\}$);
                S.arrive_time=time_p_i;
                S.left_time=time_p_j;
                SP.append(S);
                break;
            end
        end
        $j = j+1$;
    end
    $i = j$;
end
return SP

Algorithm 1: Pseudocode for SPs detection

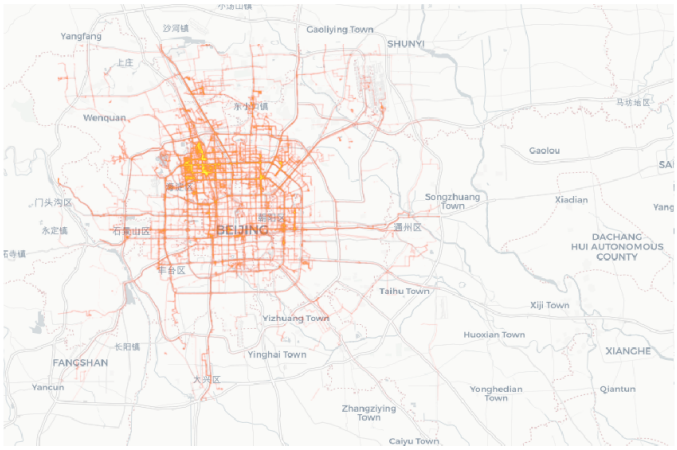

Figure 3.18: Map with the trajectories of Slot 3 in relation to the first experiments.

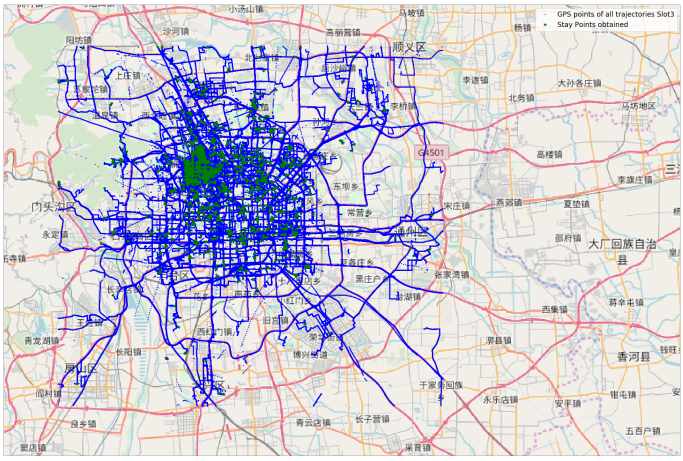

Figure 3.19: StayPoints of the trajectories on Slot 3 in relation to the first experiments.

together SPs of different users (at least 10), and checked if, in different time slots, the users that previously had a common POI move together to another one. We applied DBSCAN to the SPs obtained as it works well with large geographical dataset and likewise can be adapted for any distance functions. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular unsupervised learning method, proposed in 1996 [117], has been used in model building and machine learning algorithms.

The advantages of DBSCAN are as follows:

- it is very good for separating clusters of high density versus clusters of low density within a given dataset;
- unlike K-means, DBSCAN does not require the user to specify the number of clusters to be generated;
- DBSCAN can find any shape of clusters, i.e. the cluster doesn't have to be circular;
- DBSCAN can identify outliers.

The goal of this algorithm is to identify dense regions, which can be measured by the number of objects close to a given point. Two important parameters are required for DBSCAN: Epsilon ("Eps") and minimum points ("MinPts"). The parameter Eps defines the radius of neighbourhood around a point x. It is called the EPS-neighbourhood of x. The parameter MinPts is the minimum number of neighbours within "Eps" radius.

The last step of our work was to filter the POIs detected according to popularity. We considered only POIs with a number of users greater than 8 or 10 (called Popular POIs), in order to understand users interaction and similarity. E.g. for time Slot 3 we obtained 9 POIs shared by a minimum of 11 individuals to a maximum of 80 individuals (see Figure 3.21).

3.2.2 Results of Experiments

The experiments confirm that users stop in the same areas for some common reasons, such as visiting a tourist attraction or taking advantage of the same service and remain in certain areas in common time slots. Additionally, these tests reveal that people move together from one POI to another. The approach has been tested against several datasets (see all details in Table 3.6):

1) GEOLIFE TRAJECTORIES DATASET: we have acquired the data from the database Geolife which contains about 18 thousand trajectories with a total distance of 1,292,951 kilometers and a total duration of 50,176 hours (see Section 2.3.3). Most of the trajectories were logged in a dense representation, e.g. every $1 \sim 5$ seconds or every $5 \sim 10$

meters per point. For our research we have chosen the range of longitude and latitude of $[116.1, 39.7] \times [116.7, 40.13]$ (Beijing metropolitan area of 51 kilometers per 48 kilometers, see Figure 3.17). The trajectories of this dataset have been selected on 6 time slots of 4 hours each. Slot1 = [00:00:00, 03:59:59], Slot2 = [04:00:00, 07:59:59] and so on. The algorithms for SPs and POIs detection were started on each time interval. For the first phase the TimeTrh was chosen equal to 20 minutes, the DistThr (200 m) was unchanged instead. The execution time of the SPs detection algorithm for 100 trajectories was about 16 minutes. The clustering algorithm DBSCAN determined clusters for each time slot, with min_points equal to 10 or 15 and eps from a minimum of 200 meters to a maximum of 400 meters. The execution times were from 240 ms to 1.44s. On average 20 clusters (see Table 3.5) have been found for every time slot (120 POIs in total, for a minimum of 9 POIs to a maximum to 29) and that represent significant places for users, i.e the centroids of these clusters are POIs. E.g. for time Slot 3 with the minimum number of SPs necessary to make a cluster as 15 and the eps equal to 200 meters, 29 clusters have been, hence 29 POIs (see Figure 3.20) it has considered Popular POIs with a number of users greater than 10, for time Slot 3 it has obtained 9 POIs shared by a minimum of 11 individuals to a maximum of 80 individuals. The total of Popular POIs on all 6 time slots was 36; looking for the most distant pairs of points they have 8 km of longitude difference, 25 km of latitude difference on this area [39.91, 116.263] x [116.368, 40.128]. These places of interest in question started from the north in Yangyang Paradise (amusement park), up to the Cultural Palace of Nationalities in Fuxingmen Inner Street in south, crossing Changping District, Haidian District with Tsinghua Park, Beijing Shi and Zhongguancun. So, for our data, the clustering algorithm DBSCAN has determined clusters for all SPs. We set MinPts equal to 10 or 15 and Eps from a minimum of 200 meters to a maximum of 400 meters. We obtained on average 20 clusters for every time slot (see Table 3.5) that represent significant places for users, i.e. the centroids of these clusters are POIs. E.g. when we considered time Slot 3 and we set that the minimum number of SPs necessary to make a cluster as 15 and the Eps equal to 200 meters, we obtained 29 clusters, hence 29 POIs (see Figure 3.20).

2) TAXI TRAJECTORIES DATASET: The SP detection algorithm with a distance threshold of 200 meters and a time threshold of 5 minutes was applied: 31621 SPs were obtained with an execution time of 51 minutes and 9 seconds. The identified Popular POIs can be found in the area [116.094, 39.791] x [116.608, 40.093], they are concentrated on Beijing and the pairs of more distant points are on 45 km of longitude and on 35 km of latitude. Relatively to the POIs detection in this dataset, the parameters set in the DBSCAN were eps equal to 200 meters and minimum points (SPs) equal to 8. This algorithm produced 560 clusters whose centroids are the POIs, with an execution time of 10.2 seconds. So

they have been filtered to analyze the Popular POIs, i.e. the POIs shared by at least 8 taxis: the total number of them are 257. Out of a total of 101 vehicles, the popular points obtained were visited from a minimum of 8 taxis to a maximum number of 95.

3) TRUCK TRAJECTORIES DATASET: looking for SPs in Truck Dataset the parameters DistThr = 200 meters and TimeThr = 10 minutes have been chosen. A total of 54962 Sps were identified, in a time of 5 h 1 minute and 5 seconds. In Figure 3.22 it is possible to see trajectories recorded by trucks and their SPs, they pass through these provinces: Shānxī Shěng, Shǎnxī Shěng, Gansu, Henan, Hubei, Hebei, Beijing Municipality, Hunan, Sichuan, Guizhou, Yunnan, Guangxi Zhuang Autonomous Region, Guangdong, Jiangxi, Anhui, Fujian, Zhejiang and Shanghai Municipality. The result of the SP detection algorithm are visible in Table 3.6. The choice of different values of time threshold set in the SP detection is due to the different nature of the three datasets: for taxis a reasonable time of stay is 5 minutes, for trucks 10 minutes if it has consider the bays, for Geolife dataset, which includes routes of users on foot, it have chosen DistTrhr equal to 20 minutes. These parameters have been validated by the average speed value relative to the flows found, in the vicinity of the Popular POIs. For POIs detection the spatial threshold in DBSCAN remained unchanged (eps = 200m) and the min points = 8 as in the previous case. The clustering execution time was 5.31 seconds with 1065 POIs detected. According to the minimum number of taxis (8 out of 101), the popular POIs obtained were 127: they are shared by a minimum number of 8 trucks and a maximum number of 24 trucks.

They cross the counties: Shangsi, Longzhou, Tiandong, Long'an, Mashan, Pingnan, Teng, Yunan, the prefecture cities: Chongzuo, Wúzhōu, Yunfu, the districts Jinchengjiang and Yun'an, the Luoding city and the Kunming Subdistrict. Popular POIs obtained covered the area [106.880591, 21.609655] x [113.670971, 24.685619], with 700 km of difference in longitude and 350 km of latitude difference between the two most distant pairs of points. From west to east touched the cities: Chongzuo, Nanning, Guigang, Qinzhou, Fangchenggang, Beihai, Yulin, Wuzhou, Zhaoqing, Foshan, Canton and Dongguan.

Definitely, experiments have shown that in different time slots a set of different individuals move together to the same POIs, like parks, departments of Universities, shopping centres, hostels, parking spaces, libraries, stadiums, banks, metro and bus stops. This suggests us a similarity between users. The Points Of Interests found were verified by means of Google Maps. For each dataset used, consisting of data referring different categories of people and movements, it was shown that the points of interests are compatible with the nature of the dataset and the behavior of the users who collected them. In fact, Points Of Interests have been found in service areas, supplies and spare parts sales outlets with regard to users with trucks; service buildings and companies for businessmen traveling by

Table 3.5: Results about SPs and POIs obtained

Time Slot	**Total number of**				**DBSCAN**	
	SPs	***Users***	***POIs***	**Popular *POIs***	***Eps(km)***	***MinPts***
1	2966	122	18	8	0.3	15
2	3772	124	27	8	0.25	15
3	4146	145	29	9	0.2	15
4	1899	130	24	6	0.2	15
5	751	84	13	4	0.4	10
6	545	84	9	1	0.4	10

taxi; and places of study and work for participants of the Microsoft Research Asia Geolife project.

Among the points of interest (POIs) found we have:

- *Chaofan Weiye Kejiao Bookstore* with coordinates:

  39.98405510061326, 116.3204636235443;

- *Haidian Stadium* with coordinates:

  39.987213527969644, 116.30248430595732;

- *Beijing Rural Commercial Bank Zhongguancun Branchcon* with coordinates:

  39.980016801082485, 116.30856309688643;

- *Beihang University* with coordinates:

  39.98011363182701, 116.34218061609567.

- *Satellite Building Parking Lot* with coordinates:

  39.97673497237701, 116.33137904408086.

For validating the results of our algorithm finding POIs, each discovered site was checked against Google Maps. Hence, the above list consists of actual sites, being POIs according to Google Maps, which are within a radius of 100 meters from the POIs found by our algorithm.

From our results we can conclude the following:

- it is possible to find the POIs from a set of trajectories.

- it has been verified that the POIs correspond to well-known places, e.g. restaurants, parks, etc;

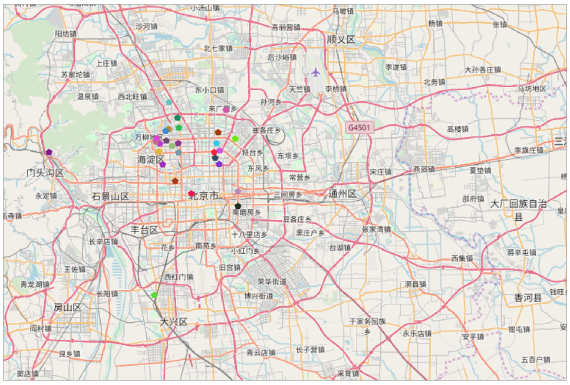

Figure 3.20: POIs obtained for the trajectories on time Slot 3.

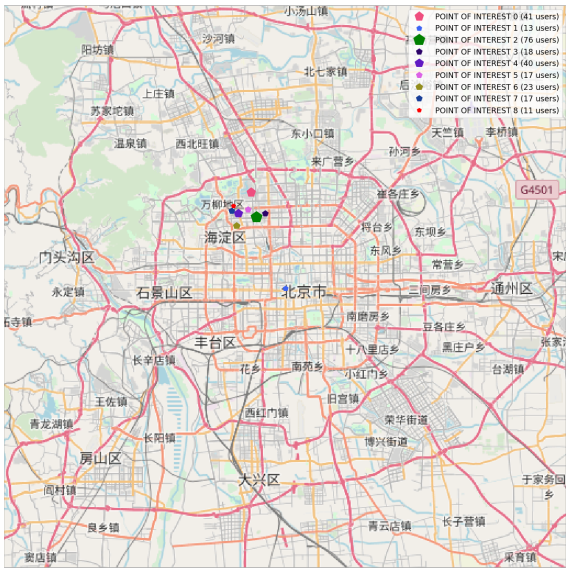

Figure 3.21: POIs with a number of users greater than 10 for the trajectories of time Slot 3.

Figure 3.22: Trajectories of Truck dataset and their detected SPs.

Table 3.6: Results in terms of StayPoints (SPs) obtained when analysing three datasets.

Dataset	DistThr (m)	TimeThr (mn)	SPs obtained
Taxi	200	5	31,621
Truck	200	10	54,962
Geolife	200	20	14,079

- the visiting hours of the POIs are uniform, therefore each POI has a time slot preferred by users. In general, the most frequent time slot is between 08:00 and 12:00;
- it has been shown that there is a correlation of people moving from one POI to another in the city.

Finally, in experiments the execution time of DBSCAN on the 6 time slots ranges from a minimum of 240 ms to a maximum of 1.44 s. The implementation uses Python 3, and the experiments were run in a host having an Intel Xeon CPU E5-2620 v3 2.40GHz, with RAM 32, 0 GB.

3.3 Attack on the user based on traceability

In this section presents a new attack to Android mobile devices able to identify their geographical location by attempting a WiFi connection, which users are unaware of. Unlike the other studies, the attack comes from a misleading app installed by the user in her device that forces WiFi to be enabled and then detect the user's position. The goal of a malicious agent is that of using an app to track the position of the device without giving any warning to the user. This is possible thanks to an appealing application, or an App Twin (e.g. an app letting the user log into two different accounts of the same service at the same time) of a well-known app, and a WiFi dataset containing SSIDs and GPS coordinates. First of all, Section 3.3.1 presents how to carry out the attack on an Android device and the needed permissions. Then, Section 3.3.2 shows how to find, for WiFi networks, SSID and GPS coordinates thanks to the use of open data.

The approach is compatible with all Android OS versions, despite the new security policies of the latest version and does not require dangerous-level permissions.

The solution to the prevention of the attack was postponed to Chapter 5, which explains the defense technique and subsequently how this technique is adaptable to the aforementioned attack. The defense mechanism firstly informs the user in real-time about the normal and dangerous-level permissions used by the application, then the user is given the possibly to block the current operation performed, by means an appropriate injected conditional code.

```
1  public void WiFiSearch() {
2      boolean inactiveWifi = false;
3      if (!isWifiEnabled()) {
4          inactiveWifi = true;
5          enableWifi();
6      }
7      List<WifiEntity> wifiEntities = database.wifiDao().getAll();
8      for (WifiEntity entity : wifiEntities) {
9          wifiConf.SSID = String.format("\"%s\"", entity.getSsid());
10         setCryptography(entity.getCryptographyType(), entity.getKey());
11         int netId = wifiManager.addNetwork(wifiConf);
12         if (tryToConnect(netId, entity.getSsid()) break;
13     }
14     if (inactiveWifi) disableWifi();
15 }
16 private boolean tryToConnect(int netId, String ssid) {
17     wifiManager.disconnect();
18     if (!wifiManager.enableNetwork(netId, true)) return false;
19     sendInfoToServer(deviceid, ssid);
20     return true;
21 }
```

Figure 3.23: Code trying to connecting to known WiFi networks.

3.3.1 Misure Case based on Sensing WiFi Networks

To carry out the attack, it is first necessary to induce the user to download the app. This can be done by creating an appealing app or an app twin or an app similar to a well-known one. Once the app has been downloaded and installed, it works in the background performing the following activities: (i) turn WiFi on, (ii) use a provided list of known networks (see next paragraph) and tries to connect to each one. Then, as soon as a connection has been successful, (iii) it sends a feedback to the server (with the device ID and the SSID of the found WiFi network). Finally, (iv) it disconnects the device and disables the WiFi (if it has been enabled by the app). The information that arrives at the server suffices to identify the user's position, since the GPS coordinates associated with the SSID are looked up on a known dataset.

Figure 3.23 shows the detail of the attack. Lines 3 to 6 turn the Android WiFi on if needed. Line 7 reads WiFi networks data from a database. On the for loop (lines 8-13), each network is configured and, *tryToConnect(int, string)* method (lines 16-21) attempts to connect to the configured network, then if successful a server is notified (line 19). Finally, if needed, the WiFi is turned off (line 14).

After the user has successfully connected to a WiFi, for performing subsequent connections attempts, the WiFi list used will have been updated by the server, which has given the highest priority to the nearest WiFi networks according to their geographical

location.

The permission-level required for the above set of steps is normal, therefore no warning will be shown to the user at runtime and the app will run undetected. The necessary permissions are as follows: CHANGE_WIFI_STATE to perform operations on the WiFi (class `WiFiManager`), i.e. to enable, disable and connect to a WiFi network; and INTERNET to send information to a server.

In order to carry on a misuse case, an app needs a set of public and private WiFi network ids. The attacker determines, a priori, the type of attack to carry on. E.g. the attacker chooses whether he wishes to monitor the overall movements of the victim, or whether to keep track of the victim on a given area. In the first case, to reveal her position the device has to be hooked to many WiFi networks, hence a list of a wide number of free WiFi networks ids is used. In the second case, the knowledge of a few WiFi networks (on the selected area) suffices, therefore the attacker can take advantage of knowing only some private WiFi networks.

For each public WiFi network, the SSID and the geographical location are needed. Public WiFi lists can be found on several websites that offer open data. E.g., *DatiOpen*[2] is one of the major Italian Open Data databases, and it offers a list of WiFi networks in Rome area, hence the data needed by our proposed scenario. In this case, the dataset has over 1600 WiFi networks and tells whether each WiFi network is public or private.

For each private WiFi network, it is necessary to know the password and the type of encryption (such data are read by means of line 7 of *WiFiSearch()* method, shown in Figure 3.23). Private WiFi networks can be chosen individually by the attacker who provides the app with data of some controlled WiFi networks, because the password (in general not public) is a fundamental element for connecting to the network.

Figure 3.24 shows geo-localised free WiFi networks in New York City. Such data are available from Open Data initiatives for many cities around the globe.

3.3.2 Distributed Interactions for the Misuse Case

This section details the interactions of a malicious app running on an Android device, detecting the presence of a WiFi network, and a server side. Figure 3.25 depicts the main steps aiming at disclosing the device location.

The *malicious app* will periodically contact the *server side*, providing its device id and (implicitly with the http request) the IP address assigned by the *Internet Service Provider* (ISP). There are several IP geolocalisation APIs and datasets the server side can use as

[2]http://www.datiopen.it

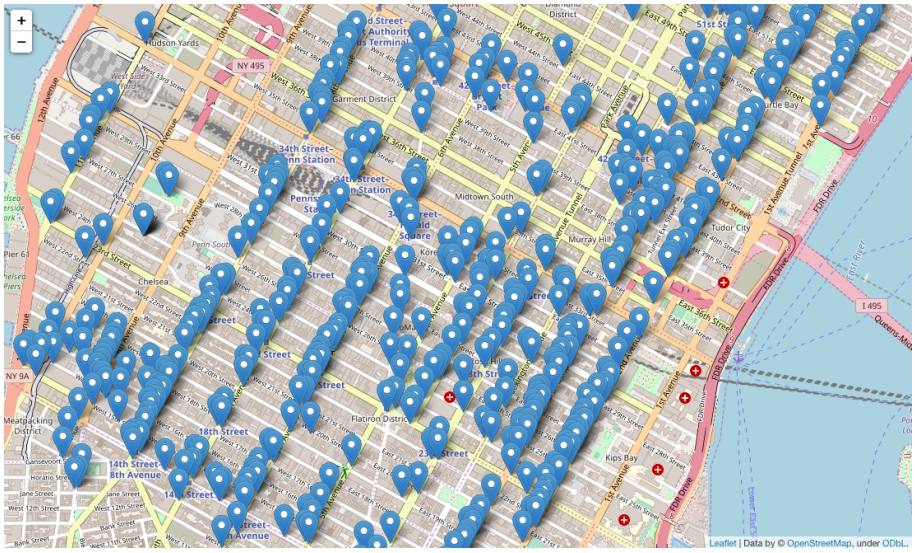

Figure 3.24: A portion of *data.cityofnewyork.us* dataset, having 3345 SSIDs available in New York City.

an *ip2gps* service (see Table 3.8). Even though IP geolocalisation is not always reliable [118], this can be used as a coarse indication of the device position, and as a starting point to filter from a large amount of SSIDs for the entire globe. Hence, knowing the IP address allows the server (at least) to exclude all the public SSIDs belonging to different countries or cities. The more widespread the servers of an ISP the more accurate the inferred position.

The geolocalised IP information, *resolved* by the *ip2gps* service, is then used to *update* the *position cache* saved on the server side. This will collect the history of all position data gathered from the device, and in turn will give a list of likely nearby *public SSIDs* to be used for the WiFi networks scanning.

With the *getScanList()* the server side will ask for the first most probable *public SSIDs* the device should be close to, according to the available information inside the *position cache*, prioritising when needed the networks located in more popular areas (e.g. city centre, shopping malls, etc.).

This reduced SSIDs list, say *[netA, netB, netC]* will then be forwarded to the device as a response to its periodic request. Then, for each SSID, the device will attempt a *password-less* connection, hence performing a WiFi scan, yet without requiring the dangerous-level permission (not needed when using the network details for an attempted connection). In case none of the connection attempts succeeded, a new list will be requested (the next most probable ones). In case a connection attempt succeeded (e.g. *netY* in Figure 3.25) this information will be sent to the server side, which will use the *ssid2gps* dataset to get a more precise gps position for the device, and then update its *position*

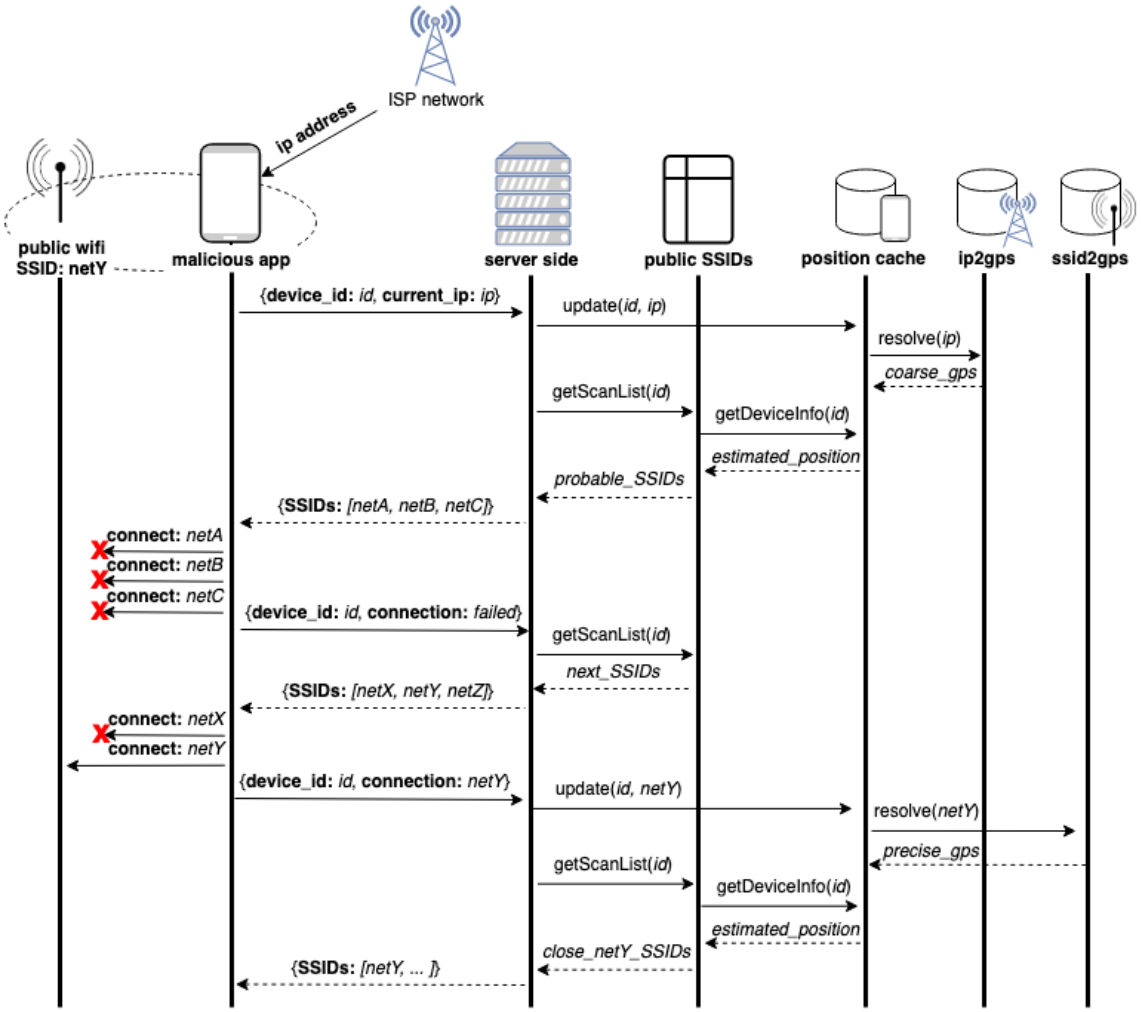

Figure 3.25: Flow diagram of the misuse case for disclosing a device location. The malicious app on the device communicates to a server side, and the server side uses two geolocalisation datasets: *ssid2gps* and *ip2gps* (see Table 3.7 and Table 3.8). Such datasets collect and translate sensed network ids into GPS coordinates, and let the server elicit the list of public SSIDs the device will be probably close to.

Table 3.7: Dataset examples of geolocalised free WiFi SSIDs

Location	Free WiFi SSIDs	Reference
Open Data Initiatives		
Rome	1,566	datiopen.it
New York City	3,345	data.cityofnewyork.us
Austin	1,575	data.austintexas.gov
Commercial Initiatives		
North America	667,275	www.wifimap.io
South America	550,733	www.wifimap.io
Europe	1,278,081	www.wifimap.io
Asia Pacific	252,070	www.wifimap.io
Middle East	1,103,220	www.wifimap.io
Africa	626,381	www.wifimap.io
Worldwide	100.000.000+	www.wifi-map.it.aptoide.com
Worldwide	600.000	www.wiffinity.com

Table 3.8: Dataset examples of geolocalised IP addresses

Dataset	IPs	Scope	Reference
ip2c.org	4,272,142,696	country	about.ip2c.org
GeoLite2	3,226,865 (blocks)	city	dev.maxmind.com
IP2Location Lite	2,959,842	latitude/longitude	lite.ip2location.com

cache. Next requests for a scan list will be answered giving the SSIDs close to the last one the device had connected to, in an attempt to constantly track the device position.

3.4 Conclusions

Nowadays, thousands of users use their mobile device to gain access to new information in relation to their geographical location. This innovation has given rise to new services, such as reading GPS coordinates in order to receive information on nearby Points Of Interest.

In this chapter we have analyzed how our information extrapolated from our device can be used for both benign and malicious purposes. We have presented two services for citizens. The first approach aims at identifying all the flows starting from a set of recorded points; the approach can give the volume of people moving in an area and find out whether there is low speed, congestion, lack of public transport means. The second approach aims to identify all points of interests common to several users starting from a series of recorded trajectories. It has been verified that the POIs correspond to well-known place, for example banks or schools. Both services were tested with real data from the Geolife project. This highlights the usefulness of data, in fact, if well structured and rich, a single dataset can allow us to implement more services.

Furthermore, we have a misuse case consisting of the steps a malicious app could perform in order to reveal the user position. The app would try to connect to known WiFi networks and when successful send this occurrence to a server. For attempting WiFi connections to known networks, an app needs to declare just normal-level permissions, hence no alert is given to users. Moreover, the server receiving network ids, when connections have been successful, can associate them with GPS locations, e.g. by using freely available datasets. This attack shows how the user can be tracked without awareness, highlighting one of the weaknesses of Android devices.

In Chapter 5 a protection tool is illustrated to guarantee the integrity of the data to the user and help him / her in understanding the data and its transmission of data on the network.

Chapter 4

Influence of the smartphone on users

Until a few years ago the transmission methods were radio, TV and billboards, nowadays there are new communication services to keep people up to date that is via smartphones. The smartphone is the bridge for sending information on places, roads, entertainment or news, through various services used by means of apps.

These services are able to passively influence the user, as the reading of news creates an imperceptible user influence. For example, the study of fakenews in the political sphere on social media was of great importance; it has been shown that users actions and political choices have been conditioned by the reading of the information. Furthermore, reading that one place is more quoted than another stimulates the user to give it greater importance.

These same services generally make (sometimes improper) use of the GPS position of the device, for advertising purposes, too. It is necessary to preserve the users' position in respect of their privacy, to avoid sending targeted advertising or other similar tools.

In this chapter, we focus on useful services supplied with lawful procedures and on third-party services that illegally inject information about smartphone, harming the user.

We present an application that sends communications to users, that is we propose an approach providing users with personalised recommendations of places of interests, such as libraries, museum, restaurants, etc, taking advantage of a multi-agent system. The approach offers a better experience by giving additional dynamic data (e.g. popularity, as number of users) to a list of Points Of Interest (POIs), and by exploring their temporal relations. The approach was designed to preserve the privacy of users, i.e. it does not reveal the position of users. The selection of points is also created based on the opinions of the users, through Sentiment Analysis techniques, this produces a more reliable service in which the common opinion prevails and not only those with greater power or money. Therefore, the analysis of feelings allows us to identify the flows of interest of ordinary people and to study and analyze user preferences, obtaining excellent business strategies. In literature, there are many tools concerned with the study of opinions, attitudes and emotions of people. So, we discuss about an innovative analysis and extraction method

that identifies people's opinions at the sentence-level. This method is based on rules derived from Italian grammar but it is easily adaptable to other languages.

Furthermore, we describe an attack on the user consisting in sending him targeted advertising by tracking his position. It shows how an app using only the permission to access Wifi networks could send some private data unknowingly from the user. This attack again emphasizes the fragility of user privacy and Android vulnerabilities. An advanced mechanism is proposed in the next chapter to shield user private data, and to selectively obscure data an app could spy.

In the first job, my contributions are as follows:

(a) it is possible for a user to immediately see the nearest recommended points by dynamically modifying the next point of the itinerary according to current needs;

(b) this research is the first approach in the tourism sector that deals with preserving users privacy in two ways: 1) using a centralized server the agents do not know each other directly to exchange information, 2) extracting information on the POIs and an approximate user location only when the user is near the POI;

(c) the integration of the recommendation system with the multi-agent system is an innovative method that encompasses the merits of both fields, allowing for constantly updated and reliable Points Of Interest;

(d) the work is the first service for tourists that integrates the concept of sentiment analysis, illustrating an advanced method based on automata.

In the second work, my contributions are as follows:

(a) unlike the other studies, the attack comes from a misleading app installed by the user in his device that forces WiFi to be enabled and then detects the users position;

(b) based on my knowledge, this research is the first concrete implementation of the attack using "access wifi network permission" and gives all details how to simulate the attack.

4.1 Point Of Interest Recommendation

POI recommendation is a service which suggests places to visit for users [85]. This section proposes an approach for POI recommendation using collaborating agents with a centralised server. The server dynamically acquires information coming from agents, which are held on the users mobile device, creating new suggestions about the next place

to visit. Common knowledge is important because, by definition, each agent can independently infer information and share it with the group.

Thanks to the collaborating agents in our architecture, the proposed approach provides users with: (i) a list of POIs, and for each point, (ii) further information based on real time data gathered from other users, which helps her choose the next destination with greater awareness. A key objective is to have this information as close as possible to real time data, and rank places according to feedback from other users, the most frequent time slots and the time spent visiting a place by other users.

4.1.1 Multi Agent System

The multi-agent system offers various opportunities and benefits. In this section we make a brief introduction of multi agents and related works in the literature.

Multi Agent System: In this approach, an agent is an application installed on an Android device that tracks the user movements around Points Of Interest (POIs), in order to suggest new places to visit [83]. A Multi-Agent System (MAS) is formed by a network of computational agents that, directly or indirectly, interact with each other [119]. In this work agents indirectly communicate among themselves as each sends data to a centralised server, hence preserving its identity. In an initial phase, each agent receives from a known server a set of POIs, computed by means of the DBSCAN algorithm [117] (as described in the following Chapter 3). Starting from such a list, each agent collects information about the device on which it runs, in relation to POIs visited by the user and sends them to the server. The server processes them to offer detailed and updated information on POIs, in order to suggest POIs recently visited and appreciated by the user.

Specifically, POIs recommendation is based on a multi-agent system performing the following steps (see Figure 4.2 and Figure 4.3).

Step 1: the server sends to agents the list of known POIs for a city. Each POI has been previously determined by the DBSCAN algorithm [117] discussed in Paragraph 3.2. Each POI has the following information:

- the most visited time slots;
- the number of agents present on the site (POI) in real time;
- a set of site feedbacks, created by a sentiment analysis algorithm that analyses the comments released by users;
- a **rating** estimated from previous information that recommends (or dismiss) the POI to the user.

There are several sentiment analysis techniques all with the common purpose of identifying the user's opinion and that can be integrated into this system [120, 121, 122, 123, 28]; some are dedicated to specific languages as in [124] which the analyzed language is Chinese. In the Section 4.2 we present a possible algorithm that uses automata designed for the Italian language that can be integrated with the recommendation system.

Step 2: thanks to the rating, the agent chooses a place of interest which the user can visit. As soon as the GPS coordinates are in a range of less than one kilometer from the coordinates of a POI, then the position will be sent to the centralised server, which can determine the number of users present. The GPS coordinates are only sent when in controlled areas in order to preserve user privacy.

Step 3: once the visit is over, the user can use the agent to issue comments on the place visited. This information will be sent to the server, and there it will be analysed using sentiment analysis algorithms, which in turn lets the server determine a score that identifies whether the POI was satisfactory for the user.

Step 4: the agent will again receive the list of POIs by adding information for a specific POI related to the site already visited based on the experiences of previous users.

In summary, points are computed by an algorithm that: (i) analyses all data coming from several mobile devices as collected by agents; (ii) computes POIs that are well-known and constantly visited by users, therefore creating the recommendation system for the users. To suggest POIs, the variables used are: the list of points of interest visited by other users, the time slots of visits and a score assigned by users as soon as their visit to the point of interest is over.

Figure 4.1 shows the overview of the recommendation system; each agent, Alice and Bob, collects information related to POIs visited by user. For each POI visited, the data forwarded to the server are: the GPS position of the device, visiting hours (through a timestamp), the name of a place, an evaluation score of the place (i.e. a number from 0 to 5). To forward such data, the user needs to authorise the application to share his GPS position. After receiving a feedback, the server processes them, creating for each point a list of information containing: time slot preferred by the users, an approximation of the number of people (that is agents) visiting the site at that time, the user feedback obtained from the average of the scores. The union of this information is used to drive the recommendation system.

Thus, the recommendation system consists of giving each user the most significant POIs for him, based on the user previous experiences. Finally, these points are sorted according to the user current distances. E.g. if there is a POI near him in which the number of people is high, and the time slot corresponds, and users have appreciated the visit (having assigned a positive score), then such a POI is suggested to the user by alerting agent Claire (see Figure 1). Otherwise, if the place has not been appreciated by other users or the time slot does not correspond, the algorithm labels that place as not recommendable. Therefore, each agent observes the users behaviour and areas they visit and offers recommendations based on current GPS positions and the experiences gained by other agents. The recommendation system is created thanks to the processing of information on the server, sent by agents.

Multi-agent integration offers a new perspective on this field. The most important properties of the agents in this proposed recommendation system are the following.

- Agents are independent: if an agent stops working the others continue their work without consequences.
- Agents are mobile: they are easily transportable thanks to their integration in the mobile application.
- Agents are reliable: given that the GPS coordinates are extrapolated directly from the device, avoiding to obtain false information.
- Agents work by preserving user privacy: there is no continuous sending of the users positions, instead the server only receives data when agents are near POIs, and the coordinates sent to the server are displaced by a bounded random amount, thus preserving the users privacy. More specifically, two important steps are taken to offer greater security to users: (i) users share the location only if they are close to a well-known POI, i.e. their position falls within 300 m from the POI); (ii) the position is processed through data masking techniques, i.e. it is saved in the central server after a displacement has been added. Having altered the position does not affect the recommendation system and guarantees better protection for the user.

In this context, security and robustness are features embedded in all the agents, working together by means of a centralised server. Thus, thanks to a system based on multi-agents, and a centralised server, it is possible to create a recommendation system based on the experience and appreciation of users, able to suggest points of interests to users, while ensuring user privacy.

Related Work: Several studies have proposed the use of multi-agent systems (MAS) [119] to a wide range of different domains. In 1998, a study described a supporting system

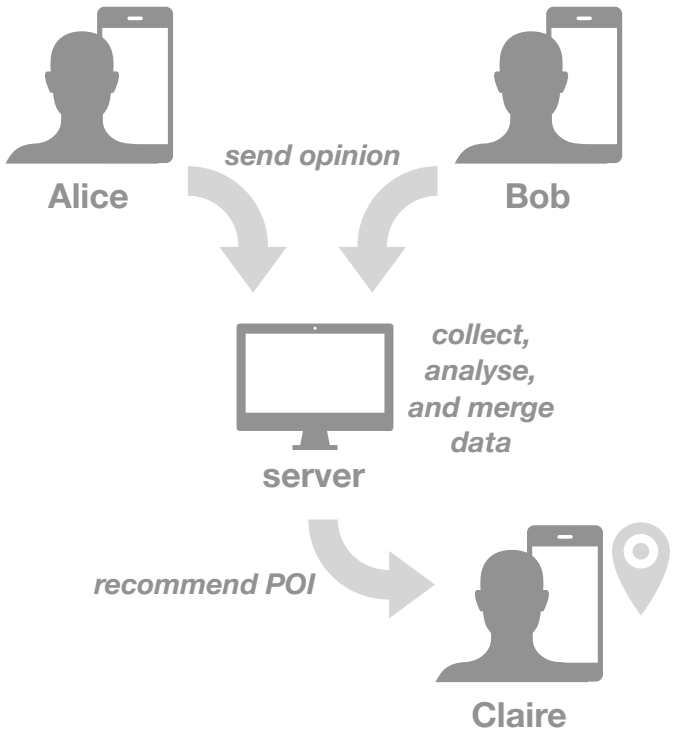

Figure 4.1: Recommendation system consisting of agents Alice and Bob gathering opinions, a server, and agent Claire receiving a recommendation.

for suggesting possible purchases during shopping based on the GPS position with the use of agents [125]. In general, there are two main approaches to MAS developments: centralized policies (CMAS) and decentralized policies (DMAS) [126].

- A centralized approach consists of taking all of the decisions in one place. In a typical CMAS, a central server collects all the relevant data that come from the different actors (that is, agents) and identifies the decisions for each agent according to the global system state. The centralized view of the system can be described by a multi-agent Markov decision process model, a good example is presented in [127].
- A decentralized approach consists of making each entity responsible for its own decision. In a typical DMAS, an agent cannot see other agents' local states and local actions, and has to decide the next local action on its own. Thus, each agent has only a partial view of the system's global state, and different agents have different partial views. A good example is in [128] whose authors propose a decentralized multi-agent decision process framework that provides the basis for a decision-theoretic study of decentralized policies.

The decentralized architecture has advantages in synchronization, reusability, scalability, and modularity [129, 130]. However, the complexity of decentralized systems is

Table 4.1: Differences between this approach (this) and other existing work

Feature	[86]	[87]	[88]	[134]	[136]	[138]	this
visiting time	v		v				v
visiting order	v	v	v		v		v
visiting duration	v	v	v		v		v
location based service					v	v	v
uses data always up to date				v	v	v	v
use economic smart devices				v	v	v	v
use multi agent system				v	v	v	v
calculates points of interest							v
preserve privacy							v

greater than that of centralized ones. Although decentralization shows obvious advantages, decentralization also has its own drawbacks, including that agents cannot predict the group behavior based only on the available local information, possible instability, and sub-optimal decisions.

Due to the importance of total knowledge, our choice fell into the first category. Moreover, the centralized server is able to filter information offering advices to users without sending their sensitive data; this preserves the user's privacy.

In recent years, the CF recommendation systems have been supported by the use of Multi-Agents. For example, in [131] the authors worked on a multi-agent system that allows for improving and optimizing the energy consumption of smart homes. Each electrical device is configured as a virtual agent. These agents work simultaneously and together to reduce consumption while ensuring user comfort, energy costs and maximum energy savings. Agent-based recommender systems have been proposed in the last years in different scenarios including the tourism [132, 133, 131, 134, 135, 136, 137]. For example, in [134], the authors present the Turist@ system based on multi-agent technology to give personalized tour attraction recommendations more effectively, highlighting the usefulness of finding points near the user. Similarly, [136] illustrates an application to better plan travel decisions based on multi-agent system. The authors of [137] propose a system that is able to produce recommendations for both individuals and groups.

Table 4.1 summarises the differences between this approach and other existing works in location recommendation. Note that, only the proposed approach makes use of gathered data to compute Points Of Interests, which therefore emerge and change while users interact with the provided system; additionally, several precautions have been taken to ensure anonymity of the user identity and their location.

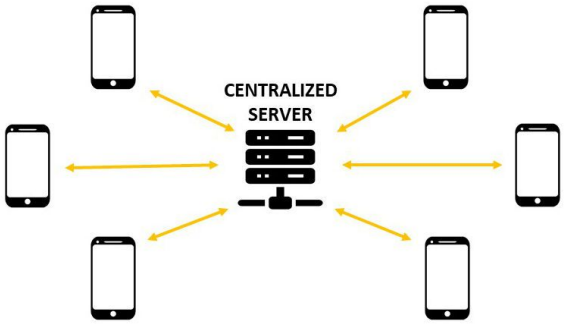

Figure 4.2: Schematic of Centralised Server. Each device identifies an agent.

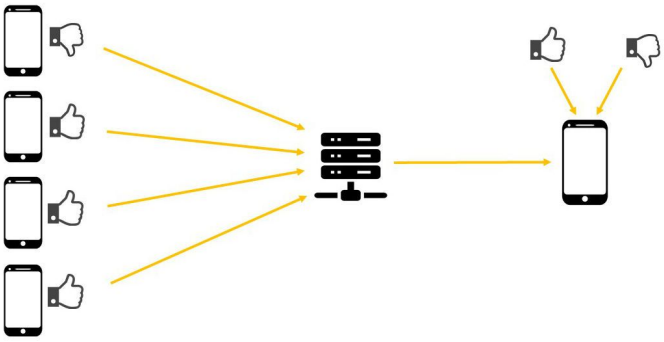

Figure 4.3: Schematic of POI recommendation. Each device sends its own information to a centralised server which processes them and suggests a new POI for agents. Using the experiences of users based on time, duration, their personal feedback and the next goal, the rating is calculated that suggests a next goal to the leading agent.

4.1.2 An Application Model

Such a cooperation can generate a benefit for groups of people who share the same interests (tourists, students, etc.). Thanks to the exchange of information between agents and servers, it is possible to define the rules for our recommendation system based on POIs (see Figure 4.3).

An application model was created for our purpose. This application extracts information from the device through the API and suggests the recommended and closest points of interest based on the GPS position of the device. In Figure 4.4 it is possible to list some suggested points with the information taken from the illustrated approach; for each POI the user can access the ratings gathered from other people comments, as well as give her comments. We can see, in Figure 4.4, four nearby points labeled as POIs in Beijing:

- *Grand View Garden*: a public park with

  GPS coordinates 39.870300, 116.356922

- *Former Residence of Mao Dun*: a museum, in particular the home of a famous writer from China with

  GPS coordinates 39.938926, 116.404114

- *Confucius Temple*: a Buddhist temple famous for its vantage point with

  GPS coordinates 39.947785, 116.412522

- *Beijing Capital International Airport*: airport area in Beijing with

  GPS coordinates 40.070854, 116.582163

For validating the results of our algorithm finding POIs, each discovered site was checked against Google Maps. Hence, the above list consists of actual sites, being POIs according to Google Maps, which are within a radius of 100 meters from the POIs found by our algorithm.

In summary, thanks to the discussed algorithm, it is possible to find a set of POIs based on gathered trajectories. In our experiments, such data were gathered by Geolife, Taxi and Truck experiments, and the set of POIs were used as a knowledge base for our recommendation system. The large amount of data was instrumental for validating our approach.

For the above proposed multi-agent system, a setting in our agent application on the smartphone allows collecting GPS coordinates. This is useful for a geographical location were there are no previously gathered trajectories. Then, we can continually extract the trajectories of users and updating both suggestions for users and POIs recommendation.

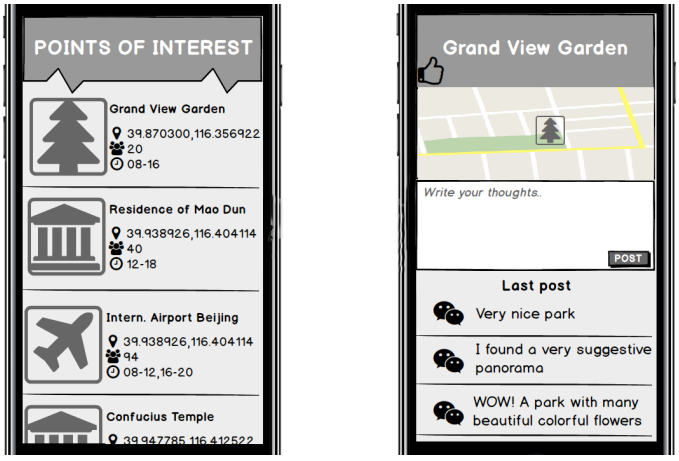

Figure 4.4: User is presented with a list of POIs and associated dynamic data.

Using such a setting, each agent releases to the server the GPS coordinates and its identity periodically, however to take into account privacy concerns, the user identity is masked, and the GPS location is randomly moved by at maximum 100 meters. Then, we obtain a trajectory which is not very precise, however still useful. As trajectories are dynamically gathered, also POIs are determined dynamically, as data arrive to the server. This is possible since our DBSCAN implementation performs well.

4.2 Sentiment Analysis

The interpretation of human thoughts is a very important field, because it allows us to achieve a variety of results, shown by several widely used applications [21]. E.g. many companies need to analyze opinions that users have expressed on some products. This complex and useful work is called Sentiment Analysis (SA) and it is the computational study of people's opinions, evaluations, attitudes and emotions [139].

Thanks to the web this field has gained more and more attention, because thousands of people tend to have collaborative and entertaining web-based relationships (e.g. being members of internet social networks and forums) as an integral part of their life. Hence, many Internet pages reflect people's thoughts, feelings and opinions. Since 2008, SA has begun attracting attention: e.g. some researchers have focused on web forums to show the usefulness of their techniques for classifying feelings in documents [140].

A large number of papers in the SA field have been published in the literature [28, 141, 140, 142, 132, 133, 131, 134, 135, 136, 137]. We propose a new approach for SA that processes text according to automata. By using three finite state automata that

implement Italian grammar rules, we are able to identify sentences that express an opinion. In particular, given a sentence we will label it as having positive, negative or neutral sentiment.

Despite the complexity of the Italian language we have obtained very good results in terms of accuracy of the classification and recall. We have implemented the approach using the MapReduce style in Java, hence speeding up execution as much as possible [71].

Thus, a presentation of the lexicon follows and in the next section we explain how the automata work, the tests performed and the study of a parallel approach.

To determine the phrase's orientation it is very important to be careful with the lexicon. Therefore, the first step is to extract the keywords. We classify words according to the following categories [143]:

1. Words of common sentiment: the terms having the same meaning regardless of the context, such as *buono* (good, delicious) and *cattivo* (bad, nasty);

2. Parts of speech (PS): terms that indicate, reinforce or deny an expression; such as in *poco attraente* (not very attractive) or *tanto attraente* (very attractive), *poco* and *tanto* are terms limiting or reinforcing a sentiment, hence classified as PS;

3. Negations: words that may change the sentiment opinion, e.g. *non* (not).

Negation is tricky because the same word accompanied by negation may change the orientation of a sentence. E.g.

Il mio nuovo pc è potente (my new pc is powerful)

versus

Il mio nuovo pc non è potente (my new pc is not powerful).

In our proposed approach words appearing in a sentence are classified as belonging to one of the following categories (see Table 4.2).

1. Verbs (V): a subset of the Italian verbs comprising all the verbs that are mostly used to express an opinion. Such a subset includes verbs: *essere* (be) and *avere* (have); and other verbs used in similar contexts: *trovare* (find), *sembrare* (look), *restare* (stay), *vedere* (see).

2. Articles (A): the list of articles, such as e.g. *il* (the).

3. Sentiments (S): a list of words related to emotions, e.g. *bello* (beautiful) and *brutto* (ugly). This list consists of two subsets: a set expressing positive sentiments, such as *carino* (nice) or *dolce* (sweet); and a set for negative sentiments, such as *cattivo* (bad) or *pesante* (heavy).

Table 4.2: Lists of sample words for each identified category, including positive and negative sentiments and adverbs, and prepositions.

positive sentiment	negative sentiment	positive adverb	negative adverb	preposition
importante (important)	inquietante (disturbing)	esatto (exact)	per nulla (in no way)	ad (for)
ampio (ample)	sporco (dirty)	pure (also)	niente (anything)	in (in)
pulito (clean)	brutto (ugly)	certamente (sure)	quasi (almost)	dalla (from)
bello (beautiful)	pesante (heavy)	assolutamente (absolutely)	purtroppo (unfortunately)	alla (at the)
migliore (best)	problema (problem)	semplicemente (simply)	neanche (neither)	agli (at the)
gentile (dear, gentle)	guerra (war)	molto (very)	nemmeno (neither)	sulle (on)
grazioso (pretty)	odioso (hateful)	sempre (always)	appena (just)	nello (in)

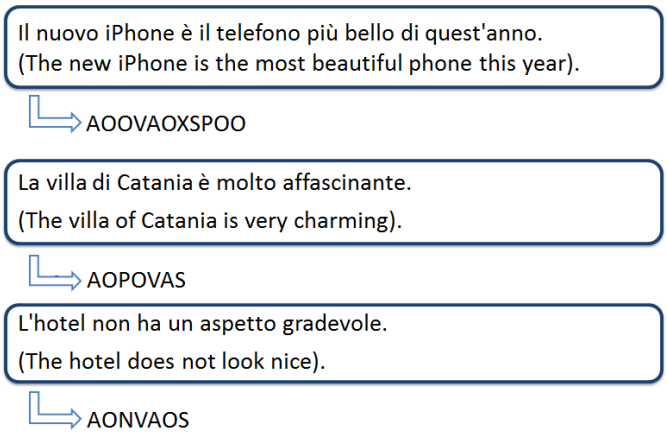

Figure 4.5: Examples of conversion of sentences based on the lexicon. For example: *Il* (the) becomes A, *nuovo* (new) becomes O, etc

4. Negations (N): it only contains the word *non* (not).

5. Adverbs (D): a list of adverbs such as *veramente* (really), *poco* (little), *molto* (a lot). This list is made up by two subsets of positive adverbs such as *molto* (very) or *davvero* (really) and negative adverbs such as *poco* (little) or *purtroppo* (unfortunately).

6. Pronoun (Y): it only contains the word *uno* (one).

7. Superlative (X): it only contains the word *più* (most).

8. Prepositions (P): a list of prepositions such as *dentro* (into) and *in* (in).

9. Other (O): the words not belonging to the above lists.

The S and D categories have two subsets because they change the meaning of the sentence and consequently its orientation.

Then, each word found in a sentence is mapped to a category. Let us analyse the sentence *Il ragazzo è amichevole* (The boy is friendly). This sentence is mapped to AOVS.

By performing slight modifications of the above sentence we can change its orientation. Indeed: *Il ragazzo è amichevole* (The boy is friendly) and *Il ragazzo è molto amichevole* (The boy is very friendly) are positive sentences. *Il ragazzo non è molto amichevole* (The boy is not very friendly) and *Il ragazzo è poco amichevole* (The boy is unfriendly) are both negative sentences.

For a correct interpretation of the sentence you must firstly convert each word in the letter of the category to which they belong (see Figure 4.5).

4.2.1 Performing Sentence Analysis

In order to analyse sentences, we have implemented three finite state automata. In particular, the algorithm uses an extension of finite-state automata, that is automata with ε-Transitions (ε-NFA).

An ε-Transition allows an automaton to change its state spontaneously, i.e. without consuming an input symbol. In diagrams, such transitions are depicted by labeling the appropriate arcs with ε.

For each automaton we have two final states, one for positive sentiment and one for negative sentiment. Each analysed sentence has a corresponding score. The score (S) starting from 0 is incremented or decremented by 1 if it is accepted by one of the three automata. Specifically, we add 1 if it is accepted by a final state that indicates a positive opinion and we subtract 1 if it is accepted by a final state that indicates a negative opinion. Finally, sentence is labelled according to the value of score S when S is greater than zero, the sentence has a positive sentiment, when S is equal to zero, the sentence has a neutral sentiment and when S is less than zero, the sentence has a negative sentiment (see Figure 4.6).

AUTOMATON 1: the first finite state automaton represents all sentences having the form:

verb + article + other + sentiment

which is a ordered sequence of the form **VAOS**, named according to the initials of the categories of words, and all the possible variations, i.e. some words can be inserted in the said sequence, without changing the expressed sentiment. For example: N**VAOS**, **VAO**A**S**, N**VAO**D**S** (see Table 4.3 for the list of all sequences).

One example of sentences having such a form is: *L'elefante è un animale molto bello* (The elephant is a beautiful animal). According to our analysis, each word is mapped into a category (see Table 4.2 and Figure 2), hence for the above sentence we have the following sequence (of categories) A**O**V**AO**D**S**.

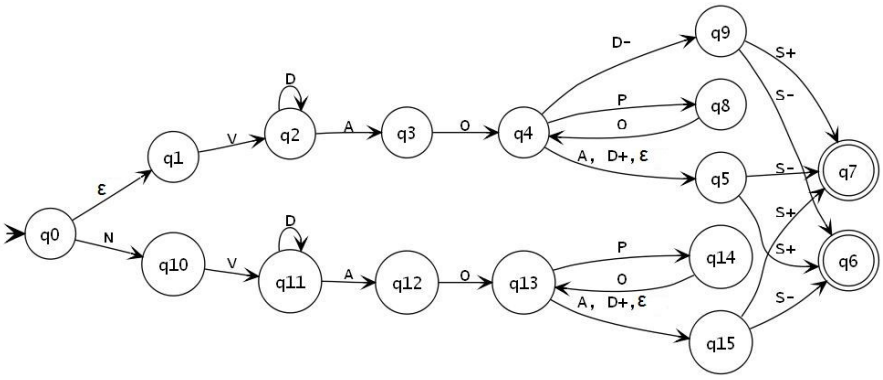

Figure 4.6: Automaton 1 recognising sentiments for sentences having form verb + article + other + sentiment

Another example for such a form is: *Il telefono ha uno schermo davvero molto resistente* (The phone has a very durable screen), which would give a sequence of categories as AO**VAO**DD**S**.

Figure 4.6 shows the automaton recognising the above sequence **VAOS** and its variations, the initial state is q0 and the final states are q7 and q6. Every sentence accepted by this automaton because ending in state q7 will be labeled with $+1$ (positive sentiment), whereas when ending in state q6 it will be labeled with -1 (negative sentiment).

The automaton in Figure 4.6 scans words of a given sentence and remains in state q0 (initial state) until a verb or a negation has been found. Each transition from a state to another is labelled with the marker of the identified word category.

AUTOMATON 2: this automaton handles sentences in which the sentiment is expressed before the name to which it refers, i.e. the name can be mentioned after the sentiment, or it can be missing. Moreover, no words expressing the names can be in between the verb and the sentiment. Therefore, Automaton 2 manages the sentences having the form:

Verb*+ Sentiment + Other*

which is a sequence **VSO** and where * indicates that it is optional. E.g. the sentence *La tua moto ha un bel colore* (your motorbike has a beautiful color) gives sequence AOO**VAS**O, which is accept by automaton 2. On the contrary, the sentence *La tua moto ha un colore bello* (your motorbike has a beautiful color) gives the sequence AOO**VAOS**, which is accept by automaton 1.

Let us take another sentence, i.e. *Il film non è stato molto interessante* (The film was not very interesting), which when processed gives sequence AON**VDS**. This is a variation of the previous **VSO** sequence, and accepted by automaton 2, since D (adverb) is a modifier for S (hence VS or VDS are similar), moreover O can be missing.

As for the previous automaton, this one (see Figure 4.7) scans words of a given sentence and remains in state q0 (initial state) until a Verb or Negation (not) has been found.

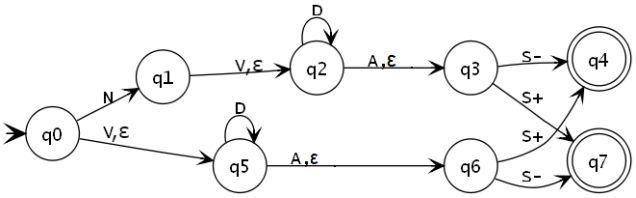

Figure 4.7: Automaton 2 recognising sentiments for sentences having form verb + sentiment + other

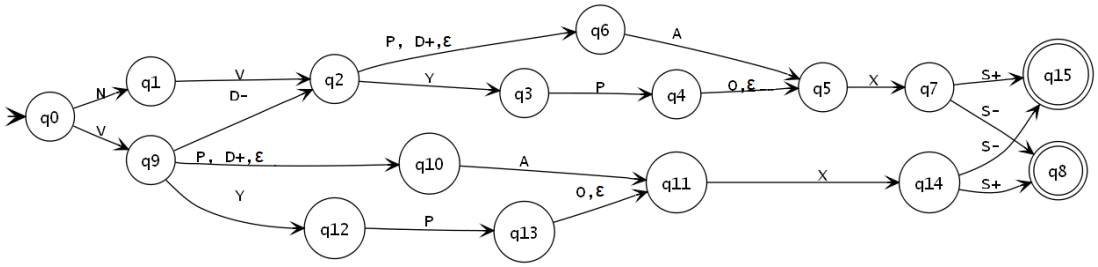

Figure 4.8: Automation 3 recognising sentiments for sentences having form verb + article/pronoun + superlative + sentiment

Then, the following words can trigger a transition towards the next states.

AUTOMATON 3: There are several studies attempting to classify a comparative sentence. In [144] authors analysed all comparative opinions (including comparative and superlative).

There is a thin difference between sentences using comparatives and ones using superlatives. In the first case the subject expresses a comparison, while in the second the subject expresses an opinion [145]. We have only chosen to consider the phrases that belong to the second category. This is represented by automaton 3.

The automaton manages the sentence with superlative excluding the phrase of comparative e.g. *La mia città è più interessante della tua* (my town is more interesting than yours). The exclusion is due to the ambiguity of the sentence itself because the sentence does not express a true opinion, but only makes a comparison. Therefore, the third automaton represents all sentences having the form:

verb + article/pronoun + superlative + sentiment

which is a sequence **VAXS** or a sequence **VAYS**, and their variations.

Figure 4.8 shows the automaton, which works analogously to the previous ones. Note that given sentence *L'auto è la più bella del mondo* (the car is the most beautiful in the world) the resulting sequence AO**VAXS** is accepted.

SEQUENCE GENERATION FROM AUTOMATA: by following all the possible paths from the initial state to the end states for each of the given automaton, we can list all the

Table 4.3: Sequences generated from the three proposed automata. The sentiment (S) and adverb (D) categories contain positive and negative words, that is: D = {positive adverbs, negative adverbs}; S = {positive sentiments, negative sentiments}.

Automaton 1	**Automaton 2**	**Automaton 3**
VAOS	VDAS	VYPOXS
NVAOS	NVDAS	NVYPOXS
VAOAS	VASO	VYPXS
NVAOAS	NVASO	NVYPXS
VDAOS	VDASO	VPAXS
NVDAOS	NVDASO	NVPAXS
VAODS	VDS	VAXS
NVAODS	NVDS	NVAXS
VAOPOAS	VDDS	VDAXS
NVAOPOAS	VDADS	NVDAXS
VAOPOS	DS	
NVAOPOS	DAS	
	S	
	NS	

possible words sequences that are accepted. Table 4.3 shows the partial lists of accepted sequences.

4.2.2 Experiments and results

The above proposed rules for finding the sentiment of a sentence have been embedded into a developed tool, which has allowed us to test the approach on unstructured text. We have considered two collections of data:

1. Collection T1: a set of about 1000 sentences that have been extrapolated from various web forums.

2. Collection T2: a set of about 800 sentences that have been extrapolated from the Yelp platform.

The selected targets are hotels and mobile phones in both data collections. The tests were performed using the described approach and Serendio algorithm. Thus, we can compute *precision* and *recall* for both approaches.

Below is an introduction of the data collection (Web Forum and Yelp) and of the Serendio algorithm.

WEB FORUM: all sentences have been randomly extrapolated from different web forums (like Tripadvisor). We target “Restaurant”, ”smartphone” and ”Hotel”. Each sentence has been manually labeled with a score (S2) which identifies the sentiment expressed:

- the score -1 identifies a “NEGATIVE” sentiment;

- the score 0 identifies a “NEUTRAL” sentiment;

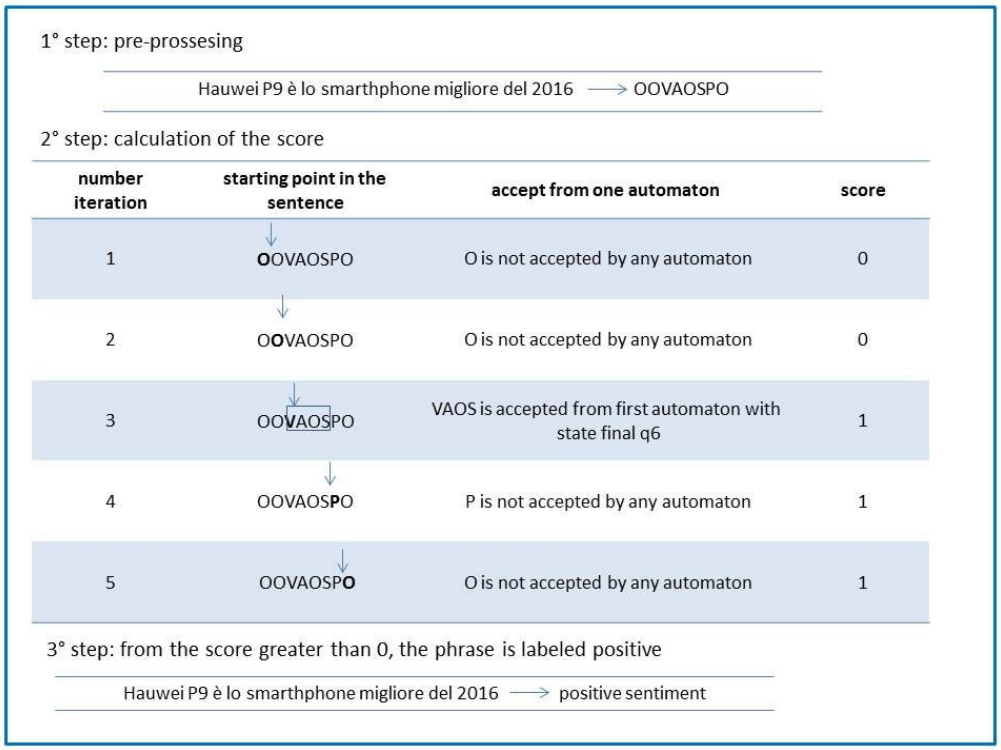

Figure 4.9: Description of the phases of the algorithm. **Input**: *Huawei P9 e' lo smartphone migliore del 2016* (Huawei P9 is the best smartphone for 2016). **Output**: positive sentiment.

- the score +1 identifies a "POSITIVE" sentiment.

For the validation of our approach, we used our algorithm to predict the sentiment expressed by comment and then compared it with the sentiment resulting from the assigned score (S2).

In other words, the algorithm was applied to each sentence and the score (S) was calculated (Table 4.4 shows some examples of analyzed sentences). If S and S2 express the same sentiment (negative, neutral or positive), we consider the algorithm's result as correct. Subsequently, accuracy and recall were calculated and discussed in Section 4.2.3.

YELP: we collected our data from Yelp platform using the Yelp's API. Yelp is a review forum that allows users to post their comments on a place, restaurant or other and to associate a number of stars ranging from 1 to 5 to give their own rating which represents the user's satisfaction, that is a low number of stars corresponds to a poor acceptance (and viceversa).

We assume that for each user comment the number of stars is an accurate measure for the sentiment express of the review, that is:

- if number of stars is less than 3 then it is considered as "NEGATIVE" sentiment;
- if number of stars is equal to 3 then it is considered as "NEUTRAL" sentiment;

Table 4.4: A sample of sentences used for sentiment analysis taken from web forums. Column 1 shows the automata that accept a sub-sequence of the sentence; column 2 shows the sentence labeled; column 3 shows the lexicon corresponding to the sentence; column 4 shows the sentiment identified by the proposed algorithm, that is positive if score is greater than zero, neutral if score is equal of zero and negative if score is less than zero; column 5 shows the state finale of automation that accept the sentence (see column 1).

A	SENTENCE	PATTERN	LABEL	Final State
1	Huawei P9 è lo smartphone migliore del 2016 (Huawei P9 is the best smartphone for 2016)	OOVAOS+PO	positive	q6
1	Lo Zenfone 3 ha un prezzo molto competitivo (The Zenfone 3 has an inexpensive price)	AOOVAOD+S+	negative	q6
1	Lo Zenfone 3 ha un prezzo poco competitivo (The Zenfone 3 has an expensive price)	AOOVAOD-S+	negative	q7
1	OnePlus ha una versione di Android molto leggera (OnePlus has a very light version of Android)	OVAOPOD+S+	positive	q6
1	OnePlus ha una versione di Android molto pesante (OnePlus has a very heavy version of Android)	OVAOPOD+S-	negative	q7
1	L'hotel Verdi non ha un aspetto grazioso (The Verdi hotel does not look pretty)	AOONVAOS+	negative	q7
2	L'iPhone è un bel telefono (The iPhone is a beautiful phone)	AOVASO	positive	q4
2	La telecamera dell'iPhone non è per nulla competitiva (The iPhone camera is by no means competitive)	AODONVD-D-S+	negative	q7
2	L'iPhone é un ottimo dispositivo sia dal punto di vista hardware che software (The iPhone is an excellent device both from hardware and software point of view)	AOVASO	positive	q4
2	L'iPhone è davvero molto bello (The iPhone is really beautiful)	AOVD+D+S+	positive	q4
3	La camera dell'hotel Verdi è la più bella del mondo (The Verdi hotel room is the most beautiful in the world)	AODOOVAXS	positive	q8
3	L'iPhone é uno dei più potenti (The iPhone is one of the most powerful)	AOVYPXP	positive	q8

- if number of stars is greater than 3 then it is considered as "POSITIVE" sentiment;

For testing, we used our method to find the sentiment expressed by user comments and then compared it with the sentiment expressed by the number of stars (Table 4.5 shows some examples of analyzed sentences). If the match is compatible, we consider the algorithm is result as correct. So in short, the number of stars was used to evaluate the stability of the algorithm. Finally, accuracy and recall were calculated and discussed in the section 4.3.

SERENDIO APPROACH: We have compared our algorithm with an approach developed by the Serendio team. Serendio is a simple and practical lexicon based approach to SA [123]. The approach extracts and analyzes sentiments for a given product and feature set.

In Serendio approach the lexicon is created through the following categories:

- common or default sentiment words (positive and negative sentiment words that have the same sentiment value in different domains, i.e. *"good"*);
- negation words (words which reverse the polarity of sentiment; i.e. *"not"*);

Table 4.5: A sample of sentences used for sentiment analysis with Yelp. Column 1 shows the sentence labeled (post); column 2 shows number of stars associated with the post; column 3 shows the score of our algorithm and column 4 shows if there is compatibility between our result and the number of associated stars.

POST	NR STARS	SCORE	LABEL
Un locale davvero bello. Perfetto per le cenette romantiche. L'ambiente è raffinato, ma i proprietari sanno metterti a tuo agio. Cibo ottimo e presentato in.. (A really nice place. Perfect for romantic dinners. The environment is refined, but the owners know how to put you at ease. Great food and presented in ..)	4	1	Match compatible
Esperienza pessima. Scegliamo questo ristorante per la notte di capodanno. Prezzo: 80 a persona, una cifra non bassa e che aveva creato in noi aspettative.. (Bad experience. We choose this restaurant for New Year's Eve. Price: 80 per person, a non-low figure that had created expectations in us..)	1	-1	Match compatible

- blind negation words (words which points out the absense or presence of some sense that is not desired in a product feature, i.e. *"needs"*);
- split words (words that are used for splitting sentences into clauses, i.e. *"but"*).

Moreover, it includes a preprocessing phase for text normalization in an appropriate form to extract the sentiments. This phase has not been implemented in our experiments because it is more appropriate for studies of tweets (not included in our sentences).

The core of the Serendio algorithm is based on the sentiment calculation; sentiment calculation (SentiSum) is the aggregation of the sum of the sentiment bearing emotions of the sentence. The algorithm processes a list of words taken from lexicon and assigns to the analyzed sentence a SentiSum as follows: extrapolating words present in the sentiment list (including positive and negative sentiments) and changes the sentiment polarity of the word if it occurs in proximity to a negative word (two word distance). For example: *"The restaurant is not good."*, the sentiment word *"good"* becomes a negative word.

So, for each word in the sentiment list (common or default sentiment word) that is present in the sentence (or clauses), if the word is a "positive sentiment" then $+1$ will be added to the SentiSum, if the word is a "negative sentiment" then -1 will be added to the SentiSum. In addition, the presence of blind negation words indicate a negation sentiment. Accordingly, initializing the SentiSum variable to zero and after doing the described steps:

- if calculated SentiSum is greater than 0, then the sentence will be tagged as "POSITIVE" sentiment;
- if calculated SentiSum is equal to 0, then the sentence will be tagged as "NEUTRAL" sentiment;

Table 4.6: Overview of my results. Comparison of the proposed and Serendio approach for SA.

Collection	Source	Algorithm	TP	FP	FN	Precision	Recall	F-Score
T1	Forum	Our approach	309	107	86	0.743	0.782	0.762
		Serendio	285	127	83	0.642	0.774	0.730
T2	Yelp	Our approach	405	90	216	0.818	0.652	0.726
		Serendio	402	92	217	0.813	0.649	0.722

- if calculated SentiSum is less than 0, then the sentence will be tagged as "NEGATIVE" sentiment.

Serendio has been implemented in the Java language. The lexicon has been manually created using the same words as our algorithm.

4.2.3 Results and Comparison of models

Our sentiment approach and corresponding tool has detected sentiment for all sentences given with a high degree of precision. Table 4.6 shows measures of *Precision*, computed as $\frac{TP}{TP+FP}$, and *Recall*, computed as $\frac{TP}{TP+FN}$, where TP is True Positive, FP is False Positive, and FN is False Negative. Low rate of recall is influenced by the restricted list of feelings (about 50 positive and about 30 negative), which can be expanded with more words.

We have compared our algorithm with an approach developed by the Serendio team. The Serendio algorithm has been implemented taking cues from [123], and executed by giving it the same list of feelings and negations used in our tests. Table 4.6 gives Precision and Recall for the two approaches. We also provide the F-score, which is $\frac{2}{1/r+1/p}$, as a measure of a test's accuracy that combines precision (p) and recall (r).

It can be seen that our proposed approach and algorithm is aimed at more precise results, thanks to the accurate proposed text analysis. We identify more opinions than the Serendio algorithm, these results are more precise and we found less false positives (FP). It is possible to see a total report of the results of T1 and T2 in Figure 4.10.

4.2.4 Parallelization approach

Given the huge amount of text sentences that are available and that are generated on the web continuously, we have implemented a parallel version of the proposed algorithm using Java version 8. Such a Java version provides a significant support inspired by the Map Reduce paradigm, and introduces an easy to use concept of parallelism for the proper use of modern multicore architectures.

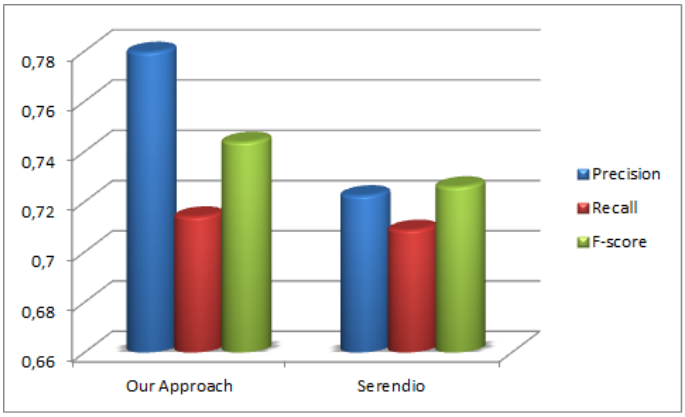

Figure 4.10: Results are an arithmetic mean of the calculated data using the T1 and T2 collections.

```
public void SentimentAnalysis() {
    map ((k, w)} -> (k, replace(w));
    reduce ((k,v)} -> (k, concatenate(v)))
    forEach (v -> checkPattern(v))
}
```

Figure 4.11: Algorithm for finding positive or negative sentiment in a sentence

In Java 8, all the classes that implement the Collection interface have a default method that returns a Stream object, which then provides methods for creating sequential or parallel execution on the streams.

In Java 8 API package, *java.util.stream* supports functional-style operations on streams of elements, such as map-reduce transformations on collections. Type Stream<T>, represents a sequence of elements and supporting sequential and parallel aggregate operations.

The implemented algorithm is given in Figure 4.11 and it splits each sentence into words (w), and then associates the relative map-reduce key (k), which identifies the position of the word in the document.

The algorithm has the following methods:

- replace(word) is the method that replaces the word with a label (or character) that identifies the category of membership for the word, such as article, sentiment, adverb, etc. (see Table 4.2);
- concatenate(letter) is the method that concatenates letters according to their key (i.e. position in the sentence);
- checkPattern(phrase) is the method that taking a sequence of word categories computes a score according to the automaton recognising it (as discussed in Section 4.2.1).

Table 4.7: Execution times for the sequential (S) and parallel (P) approach based on the size of the processed files

sentences	approach	execution time (s)
11 M	S (1 core)	287
11 M	P (8 cores)	170
16 M	S (1 core)	436
16 M	P (8 cores)	270
19 M	S (1 core)	529
19 M	P (8 cores)	325
22 M	S (1 core)	575
22 M	P (8 cores)	362
26 M	S (1 core)	696
26 M	P (8 cores)	427

Therefore, given a sentence, each word is mapped to a category, each category is concatenated to others in the same sentence, and each sequence is matched to precomputed ones that automata have generated. All the steps are performed in parallel thanks to the Java 8 stream APIs.

Table 4.7 shows the execution times for a sequential approach and a parallel approach for several millions (from 11 to 26) of sentences processed. As the table shows, we have obtained an execution time up to 48% shorter for the parallel algorithm, compared to the sequential version.

4.3 Attack and Targeted Advertising

In this last section of the chapter, we present a type of attack that injects news into the device by locating it without the user's consent. The identified attack scenario consists of two communicating sub-systems: an app running on an Android OS device and a server-side. The app aims at capturing data available on the device side, i.e. mainly the SSIDs of nearby wifi, and forward them to the server. The server analyses data and might use them to create targeted ads campaigns based on the user's current geographical location.

The identified scenario is a means to highlight the role that a set of normal permissions have for the user privacy, and to show how to make use of such permissions to find the user's geographical location.

4.3.1 Creating a Knowledge Base

An app that aims at gathering data related to SSID and geo-locations, once installed on the device would run in the background, and periodically:

Table 4.8: For the creation of our knowledge base the following methods of Android OS APIs were used, upon declaration of the related permissions; column Type identifies whether the permission level is normal (N) or dangerous (D), according to Android API level 27.

Description	Permissions	Type	Class	Method
User Identification	None	None	Settings Secure	getString(ContentResolver resolver, String name)
User Device Model	None	None	Build.Model	-
Check wifi status	ACCESS WIFI STATE	N	WifiManager	isWifiEnabled()
Turn on/off wifi	ACCESS WIFI STATE, CHANGE WIFI STATE	N	WifiManager	enableOrDisableWifi(bool)
Search for known SSID	ACCESS WIFI STATE	N	WifiManager	getScanResults()
Calculate signal level	ACCESS WIFI STATE	N	WifiManager	calculateSignalLevel(String RSSI, int level)
Describe connection status	ACCESS NETWORK STATE, INTERNET	N	NetworkInfo	isConnected()
Communicate to server	ACCESS NETWORK STATE, INTERNET	N	HttpURL-Connection	writeStream(OutputStream out), readStream(InputStream in)
Get latitude and longitude in degrees	ACCESS COARSE LOCATION or ACCESS FINE LOCATION	D	getLatitude(), getLongitude()	

- turn on the wifi (only if turned off);
- read SSIDs sensed nearby;
- send collected information to the server-side.

If the location of sensed wifi SSIDs are known, the app estimates the user's GPS position based on the SSIDs and their signal strengths. Conversely, if SSIDs are unknown, the app asks the user permission to read the current GPS location, if she accepts, the app will then send the position with the detected SSID. For the APIs reading the GPS position, the app need a dangerous level permission, hence the user has to grant such a permission.

Upon receiving data, the server-side stores, merge, and analyses them. Merging data includes an operation such as the following. SSID values which have been associated with GPS coordinates by an app on a device are shared with apps on other devices whose GPS coordinates are not available. Data analysis is performed in order to send targeted ads according to captured SSIDs, using an algorithm that filters ads conforming to geolocations.

For such an app, the needed permissions and their level are: ACCESS WIFI STATE (normal), CHANGE WIFI STATE (normal), INTERNET (normal), ACCESS NETWORK STATE (normal), ACCESS COARSE LOCATION (dangerous), ACCESS FINE LOCATION (dangerous). For more details, see Table 4.8.

The following illustrates the relevant code implemented for the app to carry out accesses to SSIDs sensed by an Android device. Figure 4.12 shows the *getWifiList()* method that returns all the wifis detected by the device (in the form of a list of strings). In detail, the type of attribute *wifi* is declared as a *WiFiManager*; the *wifiSSIDList()* method

```
1  public List<String> getWifiList() {
2      List<String> toReturn = new ArrayList<>();
3      if(wifi.isWifiEnabled()) {
4          toReturn = wifiSSIDList();
5      } else {
6          enableWifi();
7          toReturn = wifiSSIDList();
8          disableWifi();
9      }
10     return toReturn;
11 }
```

Figure 4.12: The Java code to perform the wifi scan: it enables the wifi (if turned off), looks for the available SSIDs and turns off the wifi (if it was off).

```
1  private void enableWifi() {
2      if(!wifi.isWifiEnabled()) {
3          wifi.setWifiEnabled(true);
4      }
5  }
```

Figure 4.13: The Java code to check if the wifi is active. In case of wifi off, it is activated.

calls found in lines 4 and 7 start a scan for the active wifis and get the SSID attribute of each ScanResult object returned. In line 3, it is checked whether the wifi connectivity is already active; if it is not, it will be activated and then deactivated, in lines 6 and 8, respectively. The methods for activation and deactivation are shown in Figure 4.13 and Figure 4.14. The implementation of *wifiSSIDList()* method is shown in Figure 4.15 and uses the *getScanResults()* method to have the SSIDs.

4.3.2 Architecture of a Leaking Service

Android GPS APIs can be used by an installed app to access user position. Considered the related privacy implications, for using such APIs an app has to ask the user to grant

```
1  private boolean disableWifi() {
2      if(wifi.isWifiEnabled()) {
3          wifi.setWifiEnabled(false);
4          return true;
5      }
6      return false;
7  }
```

Figure 4.14: The Java code to check if the wifi isn't active. In case of wifi on, it is deactivated.

```
private List<String> wifiSSIDList() {
    List<String> toReturn = new ArrayList<>();
    List<ScanResult> availNetworks = wifi.getScanResults();
    if (availNetworks != null &&availNetworks.size() > 0) {
        for (ScanResult sr : availNetworks) {
            toReturn.add(sr.SSID);
        }
    }
      return toReturn;
}
```

Figure 4.15: The Java code to read and return the list of available WiFi SSIDs.

a *dangerous*-level permission, i.e. the user has to explicitly allow the app to gather GPS coordinates. The request to access the device position can be motivated by some functionality, such as giving an alert for some goods to buy in a to-do list when the device position is close to a grocery shop. This will allow a server-side to leak this information also for other purposes, such as for advertisements. In fact, without further policy enforcement [41, 42], once the access to the dangerous-level permission has been granted, the user has no control on how that information will be used by the device. The only thing that the user can do is to deny the dangerous permission.

However, it is still possible to spot the user position even without asking for a dangerous-level permission, i.e. by only leveraging *normal*-level permissions a "partial" data leak, such as the list of available SSIDs, can be viable, without the user knowing about it. By asking for a SSIDs list of the nearby wifi connections, it is possible to infer the device position, then affecting user privacy. A properly designed server-side can collect and fuse leaked data with the ones sensed from other close-by devices, using an underhand crowd approach to complement partial data and improve the spotting precision.

Figure 4.16 shows the server-side architecture for an Android app able to leak device positions. In our example, the use case is a simple cloud-based *to-do-list* service accessed through an Android app installed on user devices. This app can ask access to several resources (shown in Table 4.8), such as the *device id*, the *SSIDs list* with the related *signal strength*, and *optionally* for the *GPS position*, the only one needing the dangerous permission, whose access can be denied by the user. Other data can be sensed to build a more comprehensive user profile, such as the *device model* which can give some insights about the user income, useful to target ads.

Along with legitimate behaviour, the app can periodically send the sensed data to the server-side through the *API gateway*, covered by the access to the *Trusted Services* of the given app, such as to sync the to-do-list with other owned devices. These data are collected and stored by a *Leaked Data Collector*, adding a timestamped record linked to

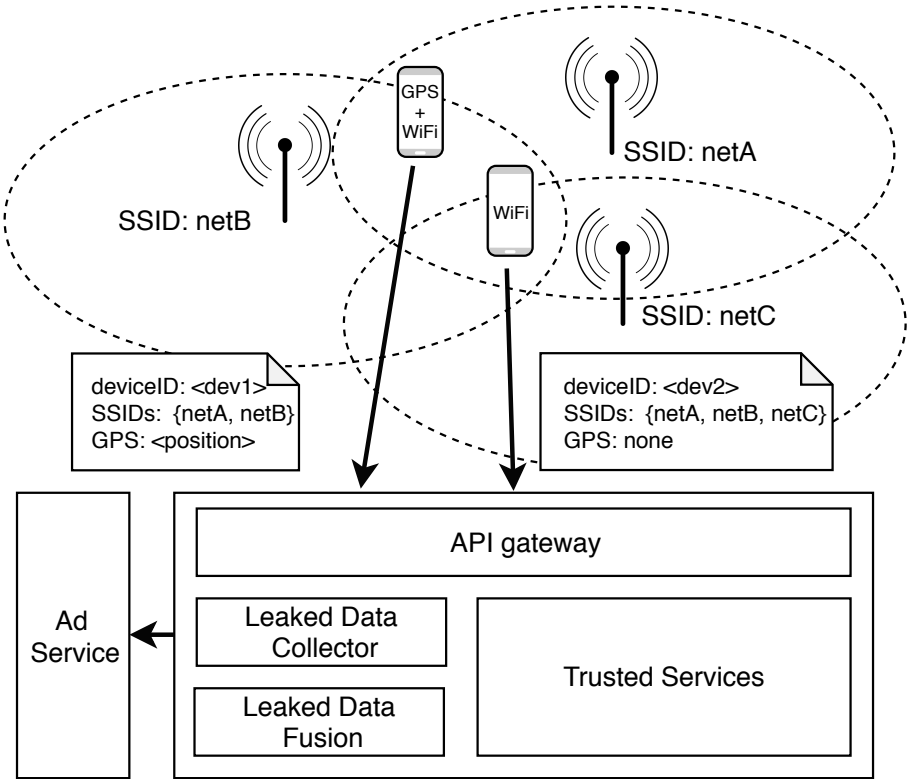

Figure 4.16: Server-side architecture: covered by the trusted service behaviour, sensitive device data can be *collected* and *fused* to refine users position. Such data can be then given to third party services (e.g. for advertising), affecting user privacy.

```
{
  androidId : "f05e06f3398c5769",
  phoneModel: "Samsung SM-G920F",
  timestamp : 10023945,
  ssidList: [
    "ssid1",
    "ssid2",
    ... ,
    "ssidN"],
  position:
  {
    lat: 111.1111,
    lng: 22.3334
  }
}
```

```
[
  {
    ssidName: "ssid1",
    position {lat: 11.333, lng: 223.44}
  },
  {
    ssidName: "ssid2",
    position {lat: 12.33223, lng: 23.47764}
  },
  ...
  {
    ssidName: "ssidN",
    position {lat: 11.3388,lng: 923.447}
  }
]
```

Figure 4.17: An example of JSON Objects sent to (on the left) and received from (on the right) the server.

specific user by the given device id. Figure 4.17 (on the left) shows an example of a JSON data record sent from the device to the server through the API gateway.

The *Leaked Data Fusion* component will fuse the collected records to build a user profile, inferring the position, when the GPS location has not been provided, by using wifi information. In the simple example depicted in Figure 4.16 there are two devices running the given to-do-list app: `dev1` and `dev2`. The former has granted the access to both the GPS and wifi information (which we call *master*), the latter only to the wifi one (which we call *slave*).

The SSIDs list sensed with normal permissions can give a clue about the device position, and given two devices the more SSIDs in common the closer they are. The fusion component can then *cluster* the devices with similar SSIDs lists, which are supposed to be close, even though their exact position is unknown (in case of a slave). Information about signal strength can also be used to weight the clustering algorithm and obtain a better clustering precision.

Then, the GPS of a master can be used to infer the position of all the other slaves in the same cluster, with a precision proportional to the SSIDs the slave has in common with the master and the related signal strength. Hence, a single close-by device providing GPS coordinates suffices to locate many other devices that have denied GPS use.

The server could also be provided with the GPS position of some known SSIDs, further improving the precision of inferred slave positions. This information can be obtained by leveraging external services like *WiGLE*[1], by a manually provided list (mapping known restaurants, parks or public buildings SSIDs), or inferred by the fusion component, in a similar way used to spot slaves. Figure 4.17 (on the right) shows a list example of known SSIDs coordinates that are used by the server to improve the spotting precision. This can also be sent to the devices for a self *GPS-less* positioning, using signal strength to estimate those SSIDs proximity.

4.4 Conclusions

Nowadays, thousands of users use their mobile device to gain access to new information in relation to their geographical location. This innovation has given rise to new services, such as reading GPS coordinates in order to receive information on nearby Points Of Interest (POIs). In this chapter we discussed an application for creating recommendations for POIs. Thanks to the points listed above, the proposed multi-agent system, exchanging

[1] https://wigle.net

information with the centralised server, has been used for the creation of a recommendation approach for POIs. The assembly of the research of the Points Of Interest of the previous Chapter with the multi-agent system offers a reliable, innovative and always up-to-date method, key concepts in this era where it is fundamental to have data updated in real time.

However, in recent years, various privacy-related attacks have been performed on mobile devices, due to their widespread use and their multiple functions. This chapter has presented a possible attack scenario for owners of Android devices, because it is possible to extract information on the users position (i.e. sensed SSIDs) without the user's authorisation, hence violating privacy. The use of these permissions to retrieve the user position is a technique extensively discussed in the literature but, unlike the others, our work implements a practical example of using this attack, explaining the various steps of the application and validating this study.

The information on the users position can be used in various ways as it allows user monitoring; an example, used in our approach, is to create a targeted advertising service.

Furthermore, to overcome such weakness we will present a defence mechanism that protects user privacy, and can alert the user id or sensed SSIDs to mask such data (technique described in the Chapter 2.1.2); such defense mechanism is discussed in Chapter 5.

Chapter 5

Advanced methods for data protection

In this chapter, we presents two approaches to safeguard users of Android devices, both protect the user from unlawful actions caused by applications installed on the device:

- Detecting Android Malware: we illustrate an approach to identify at runtime whether an operation carried out in the device is legit or not. For this, we monitor the previous behaviour of the user. An application has been created that collects data relating to the user's data traffic (such as WiFi downloaded bytes) in the background, allowing us to establish whether a new operation on the device is in line with previous ones. This gives life to a new kind of security enforcement on the device. The experiments were performed by collecting data taken from Android devices and have shown that anomalous operations can be detected with a high probability.

- Mitigating Privacy-related Risks: in order to preserve privacy, a novel and general defence solution is proposed, protecting data and resources in Android devices. Moreover, users are given the ability to configure which accesses have to be prevented and which are granted. As a proof of concept, our protection solution has been embedded in Wikipedia app, however is general and available for any app. To validate our approach, it is proposed as a solution for the attacks described in the previous chapters. Thus, in the next section it was illustrated how to use this technique in detail to be able to implement a countermeasure with each attack.

In our contribution a specific signature is not using. Consequently, by not knowing a priori the signature of the malware we can detect known malware and little known malware. In the first case a new approach has been created to identify new malware based on the profile on device owners, while in the second case the user is given the freedom of choice and control over data in a simple and intuitive way and taking into consideration also the normal permissions, allowing greater flexibility.

5.1 Detection of malicious actions

We present a new approach to safeguard the user by analysing their behaviour, collecting data from the continuous monitoring of the used device, e.g. number of packets sent, GPS, WiFi, Bluetooth, number of bytes received, number of bytes sent, etc. Our intent is to use knowledge gained on the use of a smartphone to detect anomalous operations, hence by examining previous user actions and then check whether a new operation is suspicious. An usual operation (legal transaction) is an operation that the user performs regularly and this classification (as usual) is determined by the statistics performed on user activities. An anomalous operation (illicit transaction) can derive from three scenarios: (i) operation triggered by someone other than the owner of the device (malware); (ii) operation erroneously initiated by the user; (iii) new user habits. In the third case the user would be authenticated and trigger the update of their profile.

This work aims at evaluating user habits and for this other useful information are checked, such as time and date, to understand when and where the user generally uses data connection. By collecting data and making statistics, it can then be estimated, with a certain probability, whether an action is usual or not. The approach has been tested on a portion of collected datasets offered by the authors of Device Analyzer (see Section 2.3.3) and confirmed the validity of our approach.

5.1.1 Approach based on Observations on User Activities

Everyday, users download a large amount of bytes during their Internet access for various reasons. By analysing the user's habits we can identify whether a new action is legit, i.e. according to the previous user behaviour, or anomalous with a high probability. The idea put forward is to perform a statistical analysis based on weekly/monthly reports of the user's activity, in order to identify the type of operations usually performed at runtime. E.g. consider a user that generally performs updates on Monday or Thursdays between 09.00 and 12.00 in a place with some given GPS coordinates. If it happens that she performs updates in a different time slot or in a different day of the week, this would raise an alarm, then asking the user to authenticate to complete data loading/unloading.

The proposed method consists of the following phases. Firstly, data collection: different users have different habits, hence each user has to be profiled separately. For each user, data collected relates to generated traffic, such as the number of bytes transmitted and received. This can be picked through the operating system APIs (for Android devices see Section 2.1.1). Secondly, statistics: data obtained was used to build three types of statistics, according to time intervals, weekly reports, and data connections. For each

user, one or more of the above statistics can be referred to, according to their habits. Finally, labelling requests: thanks to the previous accumulated statistics, it is possible to label a new action as usual or not. If an action is too far from the usual user actions it will be labeled as anomalous.

Case Study: the approach has been tested using a set of data provided by the Device Analyzer (DA) app. Data summarises user activities, however no sensitive data has been collected.

A) ANALYSIS AND DATA COLLECTION: during the first step, we organised and filtered data collected by the DA app. We have mainly used the following keys:

1. *system/device*: the type of device.
2. *system/model*: the model of device.
3. *net/interface/tx* and *net/interface/rx*: they are the keys that identify, respectively, the number of bytes transmitted and received.
4. *power/battery/level*: it indicates the battery level.
5. *phone/celllocation/cid* and *phone/celllocation/lac*: it indicates coordinates of GSM Base Station.
6. *wifi/connected/ssid*: it indicates the value of ssid wifi to which the device is attached.

Data used for this study involve five users. Collected data were updated on average every millisecond and concern a timeframe ranging from 4 months to a maximum of one year.

B) STATISTICS: for each user, traffic data, in relation to the weekly, daily and monthly consumption, were characterised. Moreover, wifi connection identifiers were collected and analysed. The statistics were mainly performed on two blocks of data: bytes transmitted (e.g. including *net/interface/rx* and *net/interface/tx* keys), and connected wifi (e.g. including *wifi/connected/SSID* key). These data were analysed together with the date and time in which the same data has been saved. Data concerning transmitted bytes are processed to analyse them according to the day of the week and the time slots. The time slots considered were four: 00:00-05:59; 06:00-11:59; 12:00:17:59: 18:00:24:00. Instead, data on wifi identifiers were divided by days of the week, and frequencies were computed. The median was computed on these groups in order to have a statistic of average consumption.

The statistics were made on the first two months of data available to users and tested with data available for the following months. The results have shown a degree of stability with a low error rate.

C) LABEL A NEW REQUEST: by analysing the movements collected for each user, we can create the rules, according to summarisation and statistics, that label a new action as usual or anomalous. When a new operation is carried out (by the user or in the background by the apps) our model, according to the decisions taken in the previous weeks, will assign the label. In case of unusual actions (actions labeled as anomalous), the user must authenticate himself to complete the operation. Therefore, this phase allows us to add a new level of security to the device.

5.1.2 Experiments

Experiments were performed on data offered by the authors of DA (see Chapter 2). In order to assess the performance of our proposal, we considered three types of statistics.

1. **Statistics for time slots**: the tests were divided by time slots, that is at the following time intervals: 00:00-05:59; 06:00-11:59; 12:00:17:59: 18:00:24:00. For each interval, it was reported the median value of the data (upload/download) transmitted. Data were taken for mobile internet and WiFi.

   Figure 5.1 shows the trend for time slots on a day. The x-axis gives the months in which data are measured, and the y-axis gives the amount of bytes transmitted. Plots give the median of bytes transmitted for up to 8 months. Note the stability of the median over the months. Such an analysis for time slots can be considered a good starting point to identify whether an action is usual or anomalous, because users evenly tend to consume bytes in the same time slot.

2. **Weekly statistics**: these were performed of the median of bytes transmitted according to the day of the week. The purpose is to identify whether a user generally transmits the same number of bytes based on one week or tends to download and upload data on specific days as e.g. the weekend. Data were taken for mobile internet and WiFi.

   Figure 5.2 represents the amount of data used on each day of the week. Months are on the x-axis, and the amount of bytes transmitted on the y-axis. Plots indicate the median of bytes downloaded for up to 8 months, and each represent data for one day of the week. Note that the plots become stable from the second month and roughly remain stable for the first eight months. A maximum allowable error rate is

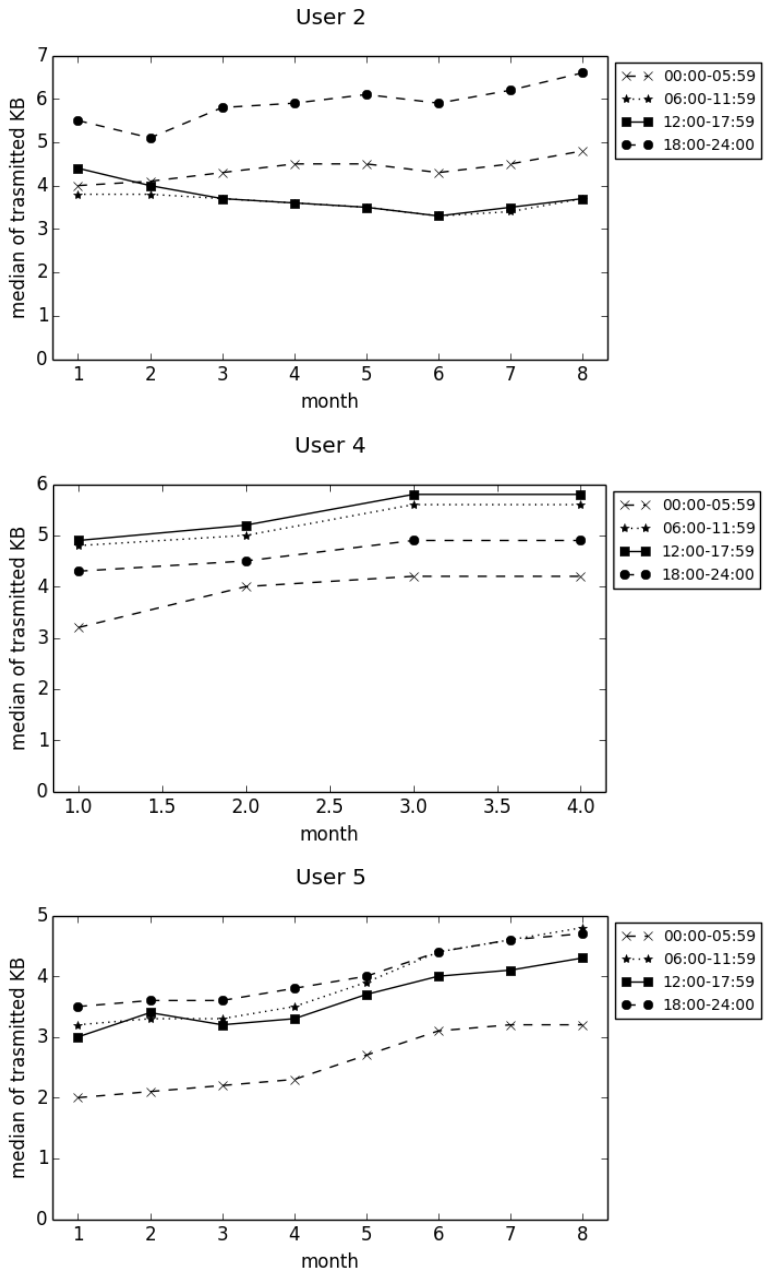

Figure 5.1: Statistics for time slots on a day. User 2 tends to consume more bytes during the interval 18:00-24:00, hence a large daily consumption could rise an alarm. User 4 has a uniform behaviour in all time slots. User 5 tends to use less data on the first time slot.

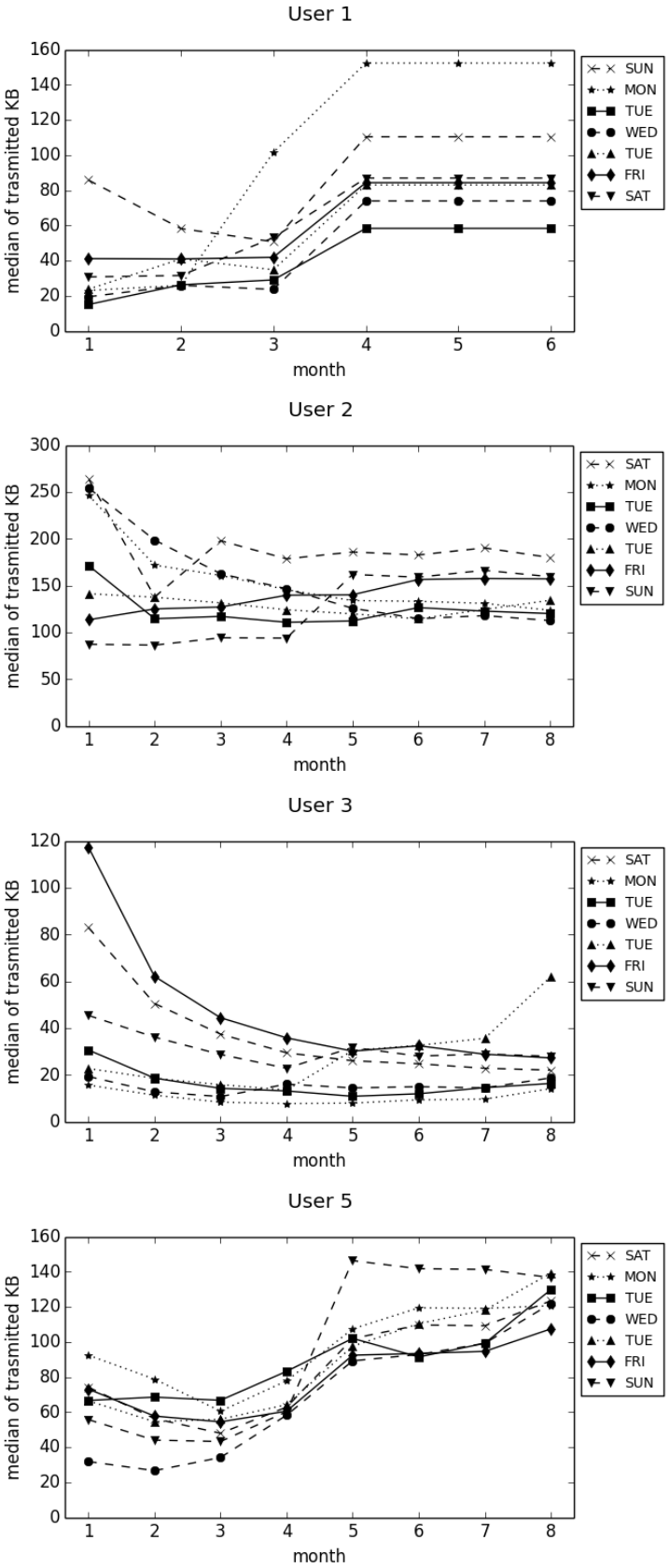

Figure 5.2: Statistics for transmitted data by day of the week.

about 20 MB. An exception, in our tests, is user 5 who, after four months, increases the number of bytes transmitted for each day of the week. We have observed that after this increase the amount of data used is stable for the following months.

3. **Statistics related to WiFi connections**: study of the transmission of bytes on a WiFi network.

   Figures 5.3 shows the WiFi connection frequency for each user. We can make the following two conclusive observations. Firstly, for every user, the highest frequency value is observed on the first month, and this is then constant for the whole period of the data available. An exception to this is user 2 (not represented in the graphs) who prefers the mobile connection, then there are insufficient data on wifi connections to perform statistics. Consequently, in general each user tends to frequently connect to the same WiFis. Secondly, some users only connect on certain days of the week.

Statistics were possible thanks to information taken from smartphones. These data were taken through specific APIs discussed in Section 2.1.1. We now describe the relationship between the above Android APIs (see Chapter 2.1.1) and the keys of the dataset used in the tests.

1. conn/wifi/state is used to verify whether the device is connected via WiFi. To receive this information, the *WifiManager* class is used, which through the *EXTRA_WIFI_STATE()* method returns a string that indicates whether WiFi is enabled, disabled, enabling, disabling, or unknown.

2. wifi/connected/SSID is used to verify which WiFi is used. This information is given by the *WifiInfo* class and its method *getSSID()* that returns the service set identifier (SSID) of the current 802.11 network.

3. net/interface/rx is used to derive the number of bytes received. We used the *TrafficStats* class, and method *getMobileRxBytes()* to read the number of bytes received.

4. net/interface/tx is used to derive the number of bytes transmitted. *TrafficStats* class and its *getMobileTxBytes()* method give the number of bytes transmitted.

5.1.3 Monitoring and responding runtime

To show the validity of the assumptions in this approach, i.e. the gathering of statistics which can be used to classify the runtime operation being performed, we created a prototype app for Android devices. The app has been implemented in Java, and properly

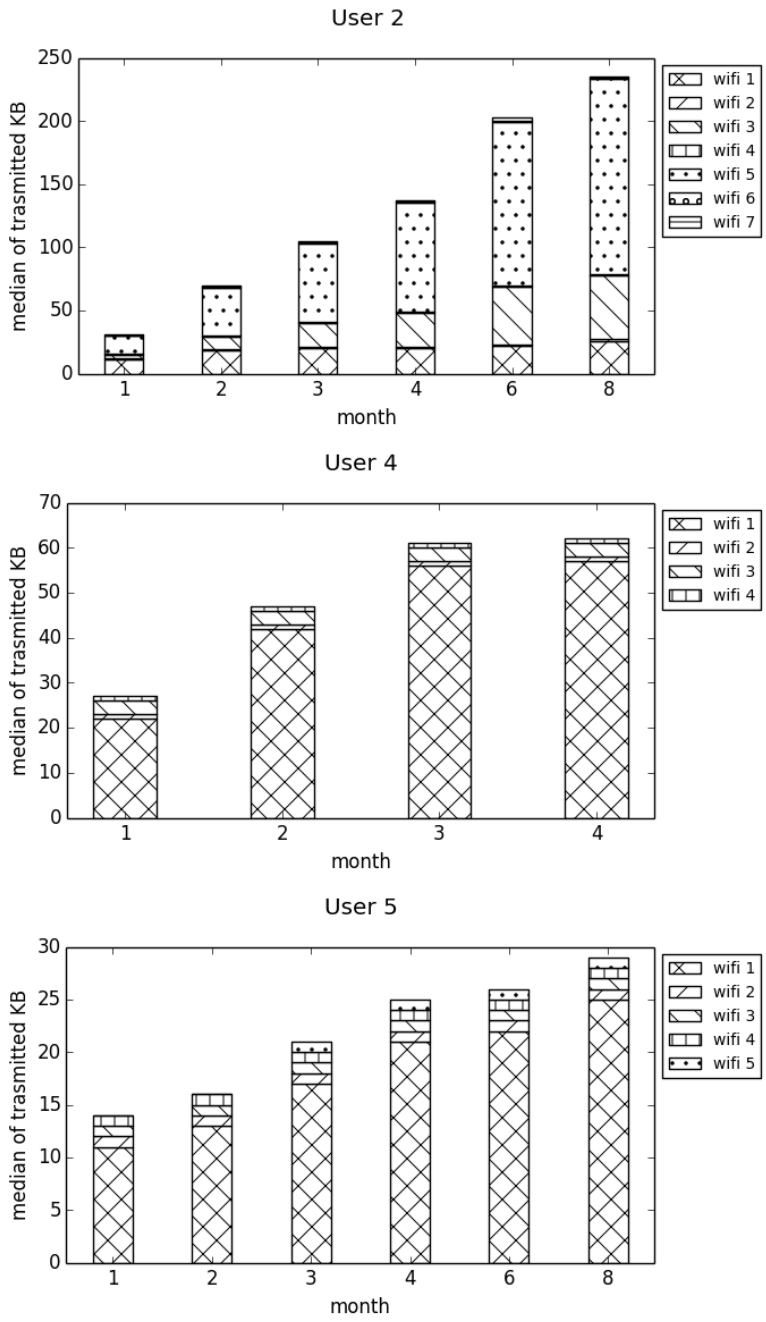

Figure 5.3: Monthly frequency of wifi connections for users. For user 2 the wifi 1, 3, 5 have the greatest frequencies since the first month. For users 4 and 5 wifi 1 frequency has a different magnitude than the others, from the first month.

Table 5.1: Permission list obtained from AndroidManifest.xml

Permission	Required by
Access Network State	the class ConnectivityManager
Access Wifi State	the class WifiManager
Access Fine Location	the class LocationManager
Bluetooth	Bluetooth Adapter
Read Contacts	to access the phone contacts
Receive Boot Completed	the broadcast action android.intent.action.Boot.Completed
Process Outgoing Calls	android.intent.action.New.Outgoing.Call
hardware location gps	Needed for API greater than 25
hardware location network	Needed for API greater than 25

detects data usage by resorting to the identification of Android APIs [146]. Once installed on the device, the app works in the background and monitors the device every time a data connection is made. This is possible thanks to the *BroadcastReceiver* component. In fact, the component listens to the entire system by triggering specific events; in our case, the *onReceive()* function is invoked, which, by using the Intent (which identifies the operation performed) will verify whether a data connection has been made.

When a connection is established, our app starts extracting various data from the device such as GPS position and byte number received/transmitted, and then later initiates the process of analysis of the data. In order to extract and study device data, the user must authorise our app to access its data, such as data from a WiFi connection. Permissions were set for this by using the *AndroidManifest.xml* file as shown in Table 5.1. When data is captured, it is sent to a server and saved to a database. This database will then be queried to make statistics.

The developed app is made of and therefore runs the following methods:

create() initialises variables and saves them in a database through *writeData()* and *manageData()* methods that are respectively located in DB and in Manager class.

start() about every one second updates the values of variables through Android APIs using different methods (e.g. *readTxFromDevice()*) that are located in Capture data class and launches, via Manager class, two methods. *saveAndAnalyzer()*: save data and update value for statistics via Statistics class; as described above, it starts three type of statistics: statistics for time slots (through *forTimeSlot()* method), weekly statistics (through *weekly()* method) and statistics related to wifi connections (through *relatedToWifi()* method). *checkInRuntime()*: thanks to the collected data and related statistics using the *matchData()* method in the Statistics class for each new action returns true if it is labeled as benign or false. In case of false output, the method starts the warning using the *launchAlert()* method

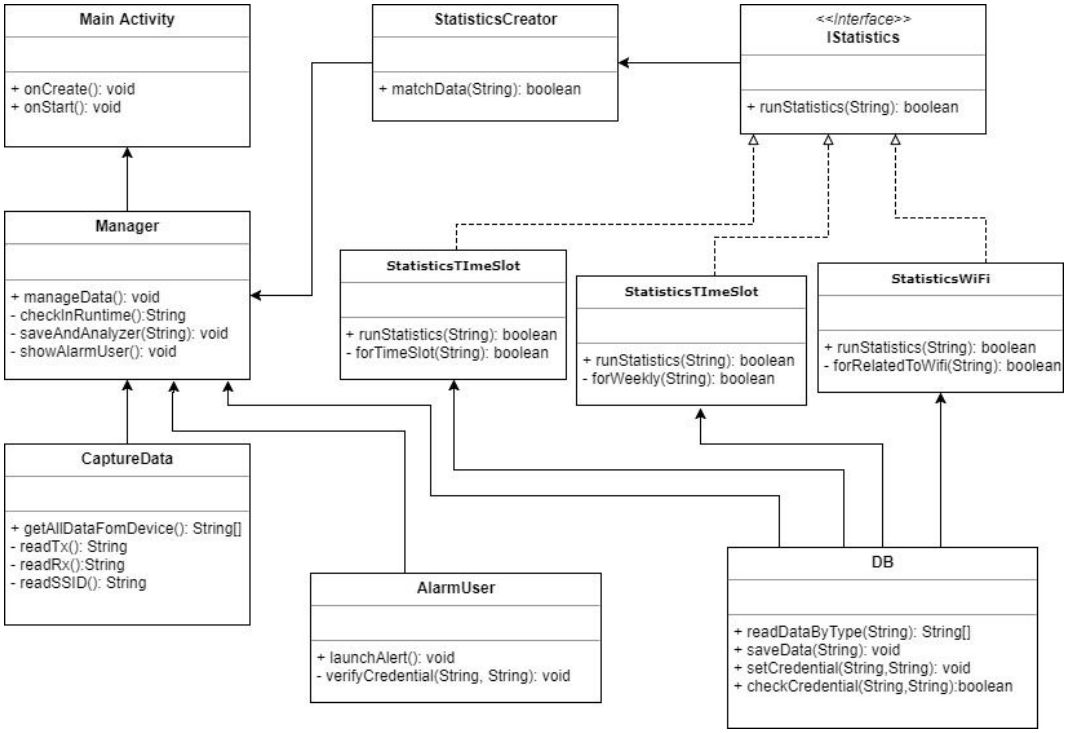

Figure 5.4: The UML diagram illustrates the classes implemented to monitor any illegal application operations using three types of statistics (based on time slot, weekly intervals and SSID WiFi).

and asks the user to identify himself with login and password (*verifyCredential()*). The last two methods are implemented by the Authentication class.

The application needs some time for creating user statistics, after that it can work through the statistics collected. Thanks to the *intervene()* method, an alert can be launched to the user at runtime, indeed e.g. if data downloaded is excessive in relation to statistics, the program will ask the user to authenticate. To continuously retrieve runtime data, a thread has been created that requires updating data every second, allowing to have a satisfactory control of the actions made. The execution time varies and depend on: the data already held on the DB (then the amount of data to be analysed), the type of device connection, the type of device.

The complete class diagram is shown in Figure 5.4 and the interaction between the various methods is described in Figure 5.5.

5.2 Mitigation of user information leakage

In this section, we propose a way to embed a defence mechanism inside the workings of an Android device. Such a proposed defence mechanism is general and powerful enough

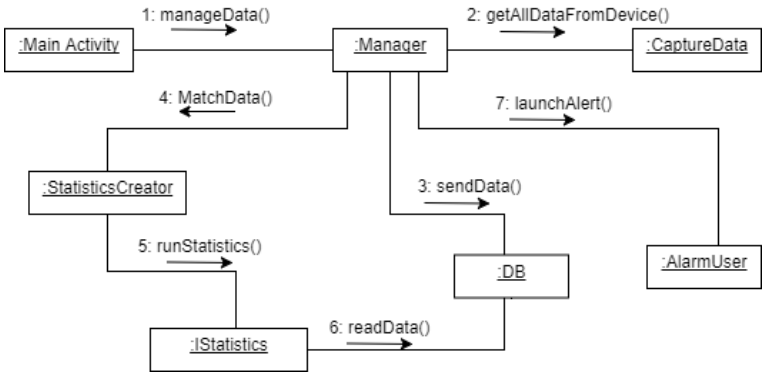

Figure 5.5: The UML collaboration diagram represents the temporal order of execution of the main methods and the collaboration between the entities.

to block unwanted accesses to sensitive data, and to let the user choose the configuration of accesses to resources. The proposed solution intercepts calls to selected APIs whose access could be not desired. Then, according to user preferences, access could be blocked or could be simply notified on the display. Interception of API calls from an app can be provided by means of aspect-oriented technology, or by app bytecode rewriting.
We have built several tests on top of Wikipedia app, however our solution is general and shows the ability of the proposed approach to give alerts, record logs, or block calls to APIs that we want to protect. Users are given an additional configuration panel to express their preferences.

This work becomes useful for the risks to which the user is exposed, discussed in the Chapter 2. In fact, in the section we discuss how normal permissions can damage the user and his privacy and how his data can be extrapolated without explicit requests to the user.

5.2.1 Defense Mechanism based on the analysis of apps

As described above, the proposed solution intercepts calls to selected APIs whose access could be not desired. Then, according to user preferences, access could be blocked or could be simply notified on the display. Interception of API calls from an app can be provided by means of aspect-oriented technology, or by app bytecode rewriting.
The aim of the proposed approach is to provide to an Android OS user: (i) additional defence and (ii) clues, against apps accessing resources usually shielded by the normal permissions level. Then, the running app is prevented from accessing some sensitive data, or sending such data over the network, and the user is given the possibility to grant or deny access to single apps, or requests, anytime, by an additional control panel designed for normal and dangerous permissions. Thanks to our approach, users are given valuable transparency on the use of their data by apps, and have a maximum comfort in

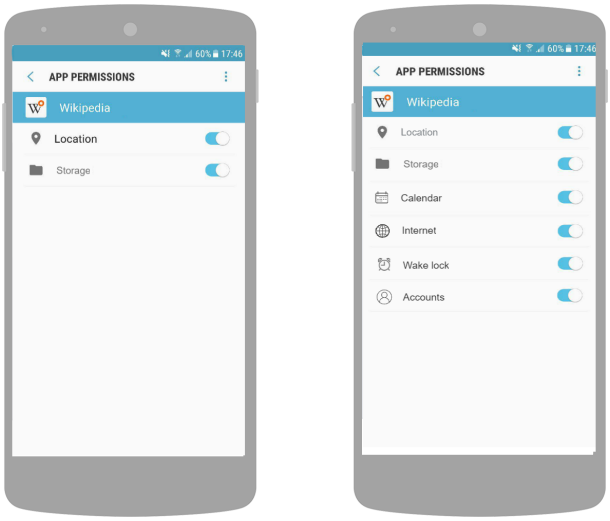

Figure 5.6: Users can choose to inhibit or allow an app to use resources managed by normal or dangerous permissions.

the use of apps. Transparency is achieved by sending configurable alerts at runtime to the user. Such alerts trace the information the app is requesting and which permissions is using. Alerts consist of messages that have to be highly visible, though small in size, so as that they do not affect the usability of the app, and offer additional clues. A tool has been developed to find and modify the behaviour of all API calls to Android libraries (under the dangerous and normal permissions) applicable to any version of Android. Such a tool monitors the use of permissions, and gives us at runtime the vision of which API the app is calling. The tool uses aspect-oriented programming for recording, alerting and checking configurations, at the moment an app makes a call to a guarded API. Let's analyze the various steps below:

A. Setting grants in a fine-grained fashion: The user is provided with a control panel letting her set which apps are granted access to resources. Such a panel includes all resources that need dangerous permissions and a subset of resources needing normal permissions. The selected normal permissions belong to those considered as data loss contributors (see Table I in Chapter 2). For each app, by using the ON/OFF switches, the user can grant or revoke the use of the various permissions (see Figure 5.6). The use of APIs related to granted permissions will be recorded and will provide alert messages to the user, as described in next section. For the revoked permissions, according to the required service, and for the correct functioning of the app, corresponding APIs will not be used, and substituted by default values.

Let us suppose that the user wants to see a map, then the app forces her to give consent on the reading of her position, it would be possible to obscure the GPS coordinates

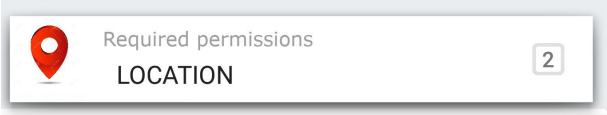

Figure 5.7: The user is shown an alert panel displaying the use of some resource by the app. This refers to the authorization of the location group.

by adding a displacement to the real coordinates. Alternatively, if an app reads the Wifi information, an altered name can be given by randomly changing letters. E.g. WLUCTU-NICT would become WAUATANACA by simply replacing the letters with even positions with the letter A. Other permissions need not be obscured by means of default values, and instead can be used according to the user setting, i.e. used only when authorisation is granted. An example of such permissions is normal permission WAKE LOCK, which if disabled prevents an app from awakening the device, hence methods of WakeLock class are simply not called.

B. Alerting Users: The provided mechanism for alerting users, firstly listens to the permissions accessed by an app, and, every time an authorisation is requested, sends a warning message to the user (see Figure 5.7).

The alert message offers the following information:

- an icon to give an immediate understanding of the permission used by the app;
- the details of the used permission;
- the number of times the app has already performed such a request.

C. Intercepting API calls: Aspect-Oriented Programming (AOP) is a powerful technique that lets developers implement a crosscutting behaviour in a modular way. Generally, an aspect consists of one or more pointcuts and one or more advices; and may contain methods and attributes, as in a class. A pointcut is a predicate that matches join point (i.e. points during the execution of a program). Pointcuts define when aspects have to intervene to change the flow of a program. An advice is associated with a pointcut expression and runs at any join point matched by the pointcut. Advices implement the logic consisting of additional operations to be performed by an aspect when reaching a given join point [147], [148]. AOP is a practical solution for adding layers of security checks in an app. We have defined an appropriate pointcut that lets us intercept all the calls that an app performs to specific APIs. E.g., the code shown below implements an example of pointcut that intercepts calls to APIs related to GSM cell data. When such calls have just been matched at runtime, an around advice intervenes to: (i) check whether permission has

been granted, (ii) log the activity, (iii) create a warning message and show it, and (iv) return a default value when permission has not been granted (method *conventionalResult()* implements the logic to find such a default value according to the given parameter representing the point of code that has been trapped. A conversion table has to be proposed for it). In our approach a warning is given to the user for both dangerous and normal permissions. Therefore, the user will be shown a dialogue window giving the count of all invocations that have been required cumulatively by the app.

```
// TraceAspectj.java
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class TraceAspect {
    private static final String GENERIC_POINTCUT_METHOD = "call(*
        android.telephony.gsm.GsmCellLocation.*)";
    @Pointcut(GENERIC_POINTCUT_METHOD)
        public void methodPointcut() {
    }
    @Around("methodPointcut()")
    public Object weaveJoinPoint(ProceedingJoinPoint joinPoint) throws
        Throwable {
        Object result;
        if (permissionGranted(joinPoint) {
            result = jp .proceed();
            alertPermission(jp. toString () ) ;
        }
        else{
         result = conventionalResult(joinPoint);
        }
        return result;
    }
}
```

In our approach a warning is given to the user for both dangerous and normal permissions.

Therefore, the user will be shown a dialogue window giving the count of all invocations that have been required cumulatively by the app.

5.2.2 Masking Techniques

The *conventionalResult()* method change option according to a on/off button of the control panel. If a button is in on, the *conventionalResult* method don't work and the app carries out its normal tasks, while if a button is in off, the *conventionalResult()* method is invoked and returns fictitious data or disable the service.
For example, if I have a Vibrator in off, *conventionalResult()* method deactivates the vibrator of the device. Below there is a list of pointcuts analyzed with behavior of the *conventionalResult()* method:

- ANDROID.PROVIDERS.CALENDARCONTRACT: to read calendar data, an application must include the READ_CALENDAR permission in its manifest file. It must include the WRITE_CALENDAR permission to delete, insert or update calendar data. In reading, the `conventionalResult()` method returns all records as empty, so an app can not read users personal information.

- ANDROID.PROVIDER.CALLLOG and ANDROID.PROVIDER.CALLLOG.CALLS: the CallLog provider contains information about placed and received calls, while the Calls class contains the recent calls. CallLogs contain information about outgoing, incoming and missed calls. You must have the READ_CALL_LOG and WRITE_CALL_LOG permissions to read and write to the call log, as well as READ_VOICEMAIL and WRITE_VOICEMAIL permissions to read and write voicemails. The `conventionalResult()` methods hides call information.

- ANDROID.LOCATION.LOCATION and ANDROID.LOCATION.LOCATIONMANAGER: together the Location class, LocationManager class provides access to the system location services. All Location API methods require ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permissions. The `conventionalResult()` method returns a likely position (close to the real one, but not exactly that).

- ANDROID.OS.VIBRATOR: the class operates on the vibrator on the device. Requires permission VIBRATE. The `conventionalResult()` method disable vibration in our device.

- ANDROID.NET.CONNECTIVITYMANAGER: this class answers queries about the state of network connectivity; it also notifies applications when network connectivity changes. Here, the `conventionalResult()` method can change WiFi information, that is, I can fool the application by claiming that WiFi connection is off. So application don't access to my connect. This allows me to isolate the connection of that single application. The application does not necessarily continue to work.

- ANDROID.NET.NETWORKINFO or ANDROID.NET.CONNECTIVITYMANAGER.NETWORKCALLBACK: android.net contains classes that help with network access, in particular android.net.NetworkInfo (or ConnectivityManager.NetworkCallback) is used for notifications about network changes. Requires ACCESS_NETWORK_STATE permissions. In this case, the `conventionalResult()` method changes network information like SSID, name, etc. For example, a WiFi name can be altered through the ramdom replacement of letters.

- ANDROID.NET.WIFI.WIFIINFO: describes the state of any WiFi connection that is active or is in the process of being set up. Requires ACCESS_WIFI_STATE, CHANGE_WIFI_STATE. The `conventionalResult()` method roughly works like the previous pointcut "ANDROID.NET.NETWORKINFO".

- ANDROID.BLUETOOTH.BLUETOOTHCLASS: represents a Bluetooth class, which describes general characteristics and capabilities of a device. The `conventionalResult()` method roughly works like the previous pointcut "ANDROID.NET.NETWORKINFO".

From 1 to 3 above, the classes need dangerous permissions and from 4 until the end classes need normal permissions.

5.2.3 Experiments

In order to demonstrate the validation of the approach, we have integrated the mechanism with Wikipedia app. Tests were carried out by combining our aspects and Wikipedia App. Wikipedia is a multilingual, web-based, free encyclopedia based on a model of openly editable and viewable content, a wiki. It is the largest and most popular general reference work on the World Wide Web and is one of the most popular websites according to Alexa rank.
The chosen app was Wikipedia in order to demonstrate the potential and versatility of

our defense solution. Despite the complexity of Wikipedia app, the overall result, after weaving it with our proposed aspects, has not compromised or slowed down operations. Thanks to our approach, it was possible to view all the accesses to sensible resources performed by the app at runtime (see Figure 5.8). In the Wikipedia app, the declared normal permissions are: ACCESS_NETWORK_STATE, INTERNET, RECEIVE_BOOT_COMPLETED, VIBRATE, WAKE_LOCK. Moreover, the declared dangerous permissions are: ACCESS_FINE_LOCATION, GET_ACCOUNTS, WRITE_EXTERNAL_STORAGE. Other permissions are: AUTHENTICATE_ACCOUNTS, MANAGE_ACCOUNTS, BIND_JOB_SERVICE (as a Signature permission). In this case, all authorisations with an explicit request from the user will be displayed in the control panel (see Figure 5.6). For our tests, the following permissions have been disabled, by using the provided control panel: ACCESS_FINE_LOCATION, VIBRATE, ACCESS_NETWORK_STATE. For the calls that access the GPS position (requiring ACCESS_FINE_LOCATION), when permission was not granted, the alternative result (given by method conventionalResult()) returns a likely position (close to the real one, but not exactly that). The app works without any error, and plots a slightly modified position on the map, and other functionalities are not affected. In other tests, we have not granted VIBRATE permissions. For Wikipedia app, the Vibrate permission is used for notifications. Therefore, when notification is needed, vibration has been bypassed by an aspect blocking the execution of notificationChannel.enableVibration(true), usually called when an alert is created. Thus, the notification alert works properly, however the device does not vibrate. Finally, when ACCESS_NETWORK_STATE has not been granted, the app only displays data in the local cache, however it shows a message warning the user of a failed connection (see Figure 5.9). Therefore, the user can read some page even when the connection to the network has been disabled. In summary, for all the tests where permissions were not given the app did not experience any error and other functionalities, rather than the ones temporarily disabled, were unaffected.

IN SUMMARY: potentially, apps send data (private or not) over the network, however users are not always informed on such a leak. We have proposed a new approach to perform an accurate screening on the use of permissions by an app, enabling users to view and check undesirable behaviours. To monitor permission use, this section proposes a prototype that fully identifies dangerous and normal permission use points at runtime. The adopted mechanisms can clearly reconstruct malicious behaviours of apps to facilitate the monitoring of data loss. The approach has been tested with Wikipedia app, however it is general and can be included in any app without having to reprogramming it, and by reusing the provided aspect code (or an extension of it).

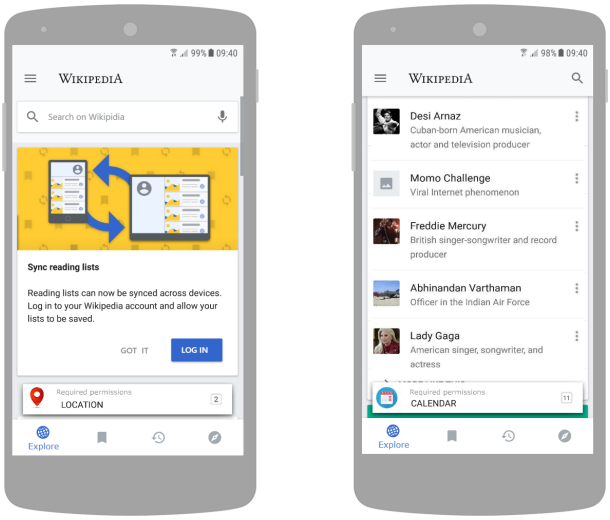

Figure 5.8: Test with Wikipedia app.

Figure 5.9: Although the device is actually connected to the internet, not granting permission ACCESS_FINE_LOCATION prevents the app from accessing the network. The app continues working with cached data only.

5.3 Countermeasures taken for attack scenarios

The measurement mechanisms described can be used for different types of attacks, both malware and external attacks (see Chapter 2.2).

In the previous chapters two types of attacks have been described. Both attacks extract sensitive information from the device without giving clear information to the owner of the device. Both techniques use WiFi connections to track the user's location.

In this section we will use the second technique described to inform the user of the possible leakage of information. Thanks to the masking techniques, the user, once informed, decided whether to hide his information to evade the attack. Therefore, among the various scenarios we present below two possible ways to use the second defense mechanism.

5.3.1 Defense for the Attack on the user based on traceability

The wide-spread availability of open WiFi networks on smart cities can be considered an advanced service for citizens. However, a device connecting to access points gives away its location. On the one hand, the access point provider could collect and analyse the ids of connecting devices, and people choose whether to connect depending on the degree of trust to the provider. On the other hand, an app running on the device could sense the presence of nearby WiFi networks, and this could have some consequences on user privacy. Based on permission levels and mechanisms proper of Android OS, in Section 3.3.1, we proposed an approach whereby an app attempting to connect to WiFi networks could reveal the presence of some known access points, thus the geographical location of the user, while she is unaware of such a feature. This is achieved without resorting to GPS readings, hence without needing dangerous-level permissions. In this section, we propose a way to counteract such a weakness in order to protect user privacy.

After having shown the hostile activities that an app on an Android device performs, and its interactions with a server side, we show a way to block the app from carry on such activities, hence protecting the user privacy.

As mentioned above, the attack involves the connection to certain WiFi networks and sending the user device ID to a server, in order to identify the victim and map the location. There are two moments when our defence mechanism can act. Both drastically reduce the data provided to the attacker. The first moment is when the app reads the device ids, the second is when the app attempts a network connection. Inspired by the above method, for bypassing or blocking calls to Android APIs, the defence mechanism consists of hardening an app modifying its code, by using Aspects Oriented Programming (AOP), to intercept a specific call and then masking the device id or the WiFi connection details.

```
@Aspect public class BlockNetAspect {
    private static final String PCUT =
       "call (* android.net.wifi.WifiManager.addNetwork(..))";

    @Pointcut(PCUT) public void blockedNetID() {}

    @Around("blockedNetID()")
    public Object blockNet(ProceedingJoinPoint jp) throws Throwable {
        if (inWhiteList(jp)) return jp.proceed();
        return -1;
    }
}
```

Figure 5.10: An aspect intercepting calls to *addNetwork()* method of WifiManager and executing instead *blockNet()* method. The call on line 9 checks whether the net id is on the white list and if so lets the call go through, otherwise a connection failure value is returned.

Assuming that the user can manage a white list, the aspect will let an app access only the WiFi networks in it. The white list, managed by a proper control panel, contains just the WiFi networks the user trusts, such as e.g. the ones at home, work, known shopping mall, etc. By trapping calls to *addNetwork()* method (lines 3 and 5 in Figure 5.10), the additional code checks whether the call to connect has an SSID contained in the white list, then connection is attempted, otherwise will return -1, not having attempted connection.

A further protection aspect can be deployed which checks whether the device ID is read by an app. Reading the device ID can be trapped by guarding calls to methods of *android.provider.Settings.Secure* class.

By checking every time the device ID is read, it is possible to mask the id, giving instead a fake one that changes periodically.

By employing the first protection aspect, the attacker will receive sparse or no data, whereas using the second protection aspect he will receive an incorrect device ID. In both cases, the attack cannot be carried on: the attacker cannot identify the user and consequently cannot monitor his movements, or identify the exact number of people who hook onto a WiFi network, and the users crossing a specific area.

5.3.2 Defense for the Attack and Targeted Advertising

Apps running on a smartphone have the possibility to gather data that can act as a fingerprint for the user. Such data comprise the ids of nearby wifi networks, features of the device where apps are running, etc. Such data can be a precious asset for offering e.g. customised transportation means, news and ads, etc. Additionally, since wifi ids can be easily associated to GPS coordinates, from the users frequent locations it is possible to

guess their home address, their shopping preferences, etc. Unfortunately, existing privacy protection mechanisms and permissions on Android OS do not suffice in preventing apps from gathering some data, such as e.g. wifi ids, which can be used as a surrogate for GPS coordinates. In Section 3.3.1, we have shown how an app using only the permission to access wifi networks could send some private data unknowingly from the user. In this section, an advanced mechanism is proposed to shield user private data, and to selectively obscure data an app could spy.

Android libraries provide several APIs to determine the device position (i.e. *geolocation*) [10]. The most obvious one is `android.location.LocationManager` class, which gives an app the current GPS coordinates, along with the ability to receive a notification when reaching a chosen area (e.g. the destination of a travel route). However, the device position could also be determined using other signal information, such as the cellular GSM/LTE state. The collaborative initiative of *OpenCellID* dataset[1] provides the GPS positions of over 36 million unique GSM Cell IDs. By knowing the list of the cell towers sensed on the device, which is given by `android.telephony.TelephonyManager` class, it is possible to geolocate the user device with a fair level of precision. This can be preferred to GPS for coverage improvement (i.e. inside buildings), or to reduce the energy consumption, avoiding the activation of the GPS receiver.

WiFi information can be used in a very similar way. *Wigle*[2] dataset lists over 565 million geolocated WiFi SSIDs around the globe. By scanning the WiFi networks to obtain the list of visible SSIDs, given by `android.net.wifi.WifiManager` class, the device position can be disclosed, again without using the GPS sensor.

An app has to declare dangerous-level permissions `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION`, before using the APIs revealing the list of nearby WiFi networks or to get the cell id, and the user will be asked to grant permissions at runtime. For the WiFi scan this was only added in Android APIs version 28[3], which is the last Android release at the date of writing. However, according to the official *Android Distribution Dashboard*[4], currently the number of Android devices using the latest Android version is less than 11% of all Android devices in use (July 2019).

The enforcement of the said dangerous-level permissions restricts WiFi scanning capabilities when looking for surrounding *possibly unknown* SSIDs. However, it is possible for an app to attempt a connection to a known access point without needing dangerous-level permissions, hence without asking the user to grant permission at runtime. By using

[1] http://wiki.opencellid.org
[2] https://wigle.net
[3] https://developer.android.com/guide/topics/connectivity/wifi-scan
[4] https://developer.android.com/about/dashboards

```
@Aspect public class TraceAspect {
    private static final String PCUT =
       "call (* android.provider.Settings.Secure.*)";

    @Pointcut(PCUT) public void blockedMethods() {}

    @Around("blockedMethods()")
    public Object weaveJP(ProceedingJoinPoint jp) throws Throwable {
        Object result = conventionalResult(jp.toString());
        viewRequestedPermission(jp.toString());
        return result;
    }
}
```

Figure 5.11: An aspect intercepting readings of the device ID. Every call to methods of Secure class is trapped by pointcut PCUT (line 3, 5) and weaveJP() method is executed. Then, the call on line 9 changes the device ID, and the call on line 10 sends an alert to the user.

such a feature and by leveraging the OpenData about free WiFi networks, the proposed approach presents a misuse case for determining the device position with a fair level of precision.

More specifically, a malicious app can be implemented to force the Android device to systematically try to connect, and suddenly disconnect, to a list of known open (i.e. with no password required), and even some password-protected, geolocated WiFi networks, in the quest for locating the user.

Open data initiatives taken by several government institutions, such as for the City of Rome[5] and New York City[6], give access to datasets about geolocated open WiFi SSIDs, e.g. to help tourists in finding a free network access near monumental buildings or historical sites. Moreover, other *WiFi Finder* apps like *WiFi Map*[7], used to spot nearby free WiFi networks while sharing newly discovered ones, often provide commercial plans for developers needing API access to the overall geolocated dataset.

Given the risk of information leakage, in this section we propose an enhanced defence mechanism that manages to protect user privacy. For the scenario illustrated above, an app can gather the device id and the SSID of close-by wifi networks and pass them to a server. This can be considered a violation of the user's privacy, since the former gives the server a surrogate for the user identity, and the latter a close approximation of her position.

To overcome such a privacy violation, the app can be automatically transformed, e.g.

[5] http://www.datiopen.it/it/opendata/Provincia_di_Roma_WiFi

[6] https://data.cityofnewyork.us/City-Government/NYC-Wi-Fi-Hotspot-Locations/yjub-udmw

[7] https://www.wifimap.io

by means of the Aspects Oriented Programming (AOP), to intercept all the calls for reading the device id.

Therefore, the call pertaining to the *User Identification* request (see Table I) is intercepted by our provided aspect and, periodically, or when a command is given by the user, the ID is altered using masking techniques, before being sent to the server (see Figure 5.11). Moreover, the user can be alerted of such an access. For example, we could simply apply the Caesar cipher [149] with a value *n* defined by the user. That is with n = 2, the ID d83c84d1176a3547 becomes f05e06f3398c5769.

In this way, even though the server has the SSIDs of close-by wifi networks, it is unaware of the real user id, hence several data analyses become not meaningful. E.g. the server cannot properly compute the user frequent locations, since the user id varies with time, or cannot profile the user because the real id is hidden by multiple (fake) ids.

Similarly, the read operation of SSIDs can be captured and the real values can be changed upon each reading operation, before the app can send them to the server. Making the values of SSIDs vary for each read operation renders the server data base useless, since a device cannot be associated with a list of known nearby wifis.

On one hand, data leaks can be very difficult to spot among all communications with a server legit from a functional point of view, hence they cannot be blocked. On the other hand, the proposed aspects selectively block private information, and could be automatically made available just before deployment by an agent handling the app market or a certification authority that aims at hardening the app.

5.4 Conclusions

Potentially, apps send data (private or not) over the network, however users are not always informed on such a transmission and, so, the apps can send confidential data to third parties without authorization.

In this last section we have presented two innovative methods based on real-time behavior of applications to detect traces of threats or danger actions. The first defense mechanism collects statistics on user behavior and it identifies whether an operation is conforms to the user's habits, creating the first user's protection. This mechanism was tested with a set of data from real Android devices, validating our thesis.

The second defence mechanism performs an accurate screening on the use of permissions by an app, enabling users to view and check undesirable behaviours. To validate our approach it was first of all tested on a famous app "Wikipedia App" and it was also used

as a countermeasure for two attacks that take information from the device without requiring the agreement to access the geographical position (for example the GPS position or information concerning the WiFi).

To monitor the user, these approaches use both normal and dangerous permissions at run time, highlighting the importance of processing normal-level permissions.

The proposed approaches can make a big contribution to the user's security, as they allow to protect the user informing him in runtime of possible data loss or loss of control of his device.

Chapter 6

Conclusions

Today, the favorite pastime of every person is to stay connected to their mobile phone to take advantage of the multiple functions or to talk with friends and relatives via social media. This disproportionate use has led to new research in the field of smartphone to analyze devices and the side effects of their abuse. In literature, there are various research on service for user, possible attacks and various countermeasures to protect the user, his data and his private life.

In this thesis, benefits and risks of data flows traveling from/to the smartphone through the network have been studied, exploiting APIs. The APIs are an important resource for Android Operating System, they allow to extract interesting data from the device such as GPS position, telephone contacts, information on the WiFi to which the device is connected, etc.

During this work, APIs were used to analyze the sending (information entered on the network) and receiving (information coming from the network) of data from smartphone; we dealt with creating services for the user, scenarios of attacks against users and advanced methods for safeguarding user.

The services are useful for users because they offer important information to citizens, such as searching for a specific place (like local or museum). In this thesis, three different services have been illustrated with the main purposes of improving the quality of public transport services and improving the management of points of interests (that is, detecting these places in large cities and creating recommendation services for users).

Services are possible thanks to a knowledge base that can be extracted from smartphones; by extrapolating trajectories of users, new knowledge can be obtained to offer innovative and always updated services. The services created are able to extrapolate many different useful information, solving problems of different nature, such as the identification of points of interest or traffic management. In literature, there are not similar works, so flexible that it is possible to be adapted to more problems. Furthermore, some of these services have been integrated with a multi-agent system, making the information always

up-to-date and more compliant with the user who uses it, giving a strong innovation. Each service has been created in respect of the user's privacy.

Subsequently, two attacks were created for Android devices. The first attack extrapolates data from smartphone processing them and silently identifying the user's geographical position. The second illustrates an alternative technique for extrapolating the GPS position of the device in order to send targeted advertisements describing the influence on the user of such news. Methods offer two different ways to monitor the victim, without being aware of it and without Android making explicit requests, since it does not use dangerous level permissions, that is, which require explicit user consent. This new monitoring techniques, tested on the latest versions of Android, highlight how there are flaws on the Android operating system, even today.

To conclude, two advanced methods for protecting user data have been presented. The first method monitors user activities, while the second monitors application requests. Both work in real time and offer instant alarm messages to the user helping them to understand the general actions of the device. In detail, the first method identifies anomalous operations, that is not customary to the user's standard behavior, while the second analyzes dangerous and normal permissions to warn the user of what information is requested by the single applications. The main features of these new techniques that can mitigate user privacy are: the ability to find malware that is always different and without knowing its signature, receiving notification messages in runtime for the user who is notified of an action potentially damaging and analyzing normal and dangerous permissions.

The methods were tested with real data taken from mobile devices or taken on the net. Due to the large amounts of extracted data, filtering and parallelism methods have been implemented to offer "clean" data and fast responses, demonstrating the feasibility of searches in real fields. Thanks to these techniques, response times have been halved by over 50 %.

Bibliography

[1] S. Allen, V. Graupera, and L. Lundrigan. “The smartphone is the new PC”. In: *Pro Smartphone Cross-Platform Development*. Springer, 2010, pp. 1–14.

[2] P. Lugtig and V. Toepoel. “The use of PCs, smartphones, and tablets in a probability-based panel survey: Effects on survey measurement error”. In: *Social Science Computer Review* 34.1 (2016), pp. 78–94.

[3] T.-M. Grønli, J. Hansen, G. Ghinea, and M. Younas. “Mobile application platform heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS”. In: *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. IEEE. 2014, pp. 635–641.

[4] P. Faruki and et al. “Android security: a survey of issues, malware penetration, and defenses”. In: *IEEE Communications surveys & tutorials* 17.2 (2015), pp. 998–1022.

[5] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. “Google android: A comprehensive security assessment”. In: *IEEE Security & Privacy* 8.2 (2010), pp. 35–44.

[6] E. Tramontana and G. Verga. “Get Spatio-Temporal Flows from GPS Data”. In: *Proceedings of IEEE International Conference on Smart Computing (SMARTCOMP)*. 2018, pp. 282–284.

[7] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez. “Next place prediction using mobility markov chains”. In: *Proc. of Workshop on Measurement, Privacy, and Mobility*. ACM. 2012.

[8] H. Huang, G. Gartner, et al. “Using trajectories for collaborative filtering-based POI recommendation.” In: *IJDMMM* 6.4 (2014), pp. 333–346.

[9] L. Huang, Q. Li, and Y. Yue. “Activity identification from GPS trajectories using spatial temporal POIs’ attractiveness”. In: *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on location based social networks*. ACM. 2010, pp. 27–30.

[10] Google. *Android Support Library*. developer.android.com/topic/libraries/support-library. 2019.

[11] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. "Permission evolution in the android ecosystem". In: *Proc. of Annual Computer Security Applications Conference*. ACM. 2012, pp. 31–40.

[12] C. Gibler, J. Crussell, J. Erickson, and H. Chen. "AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale". In: *International Conference on Trust and Trustworthy Computing*. Springer. 2012, pp. 291–307.

[13] C. Montealegre, C. R. Njuguna, M. I. Malik, P. Hannay, and I. N. McAteer. "Security vulnerabilities in android applications". In: (2018).

[14] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. "A survey of mobile malware in the wild". In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM. 2011, pp. 3–14.

[15] D. T. Wagner, A. Rice, and A. R. Beresford. "Device Analyzer: Large-scale mobile data collection". In: *ACM SIGMETRICS Performance Evaluation Review* 41.4 (2014), pp. 53–56.

[16] A. Karim, S. A. A. Shah, R. B. Salleh, M. Arif, and R. M. Noor. "Mobile botnet attacks–An emerging threat: Classification, review and open issues". In: *KSII Transactions on Internet and Information Systems (TIIS)* 9.4 (2015), pp. 1471–1492.

[17] W. Jeon, J. Kim, Y. Lee, and D. Won. "A practical analysis of smartphone security". In: *Proc. of Symposium on Human Interface*. Springer. 2011.

[18] 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*. ACM. 2012, p. 3.

[19] 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: *International conference on financial cryptography and data security*. Springer. 2012, pp. 68–79.

[20] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy. "Android permissions: a perspective combining risks and benefits". In: *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. ACM. 2012, pp. 13–22.

[21] R. Feldman. "Techniques and applications for sentiment analysis". In: *Communications of the ACM* 56.4 (2013), pp. 82–89.

[22] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma. "A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms". In: *IEEE Access* (2019).

[23] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez. "Puma: Permission usage to detect malware in android". In: *Proc. of International Joint Conference CISIS-ICEUTE-SOCO Special Sessions*. Springer. 2013, pp. 289–298.

[24] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. "Mining interesting locations and travel sequences from GPS trajectories". In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 791–800.

[25] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. "Map-matching for low-sampling-rate GPS trajectories". In: *Proc. of ACM Conf. on Advances in Geogr. Inform. Sys.* 2009.

[26] Y. Yue, B. Hu, et al. "Identifying shopping center attractiveness using taxi trajectory data". In: *Proceedings of the 2011 international workshop on Trajectory data mining and analysis*. ACM. 2011, pp. 31–36.

[27] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. ""Andromaly": a behavioral malware detection framework for android devices". In: *Journal of Intelligent Information Systems* 38.1 (2012), pp. 161–190.

[28] B. Liu. "Sentiment analysis and opinion mining". In: *Synthesis lectures on human language technologies* 5.1 (2012), pp. 1–167.

[29] C. Harrison and I. A. Donnelly. "A theory of smart cities". In: *Proceedings of the 55th Annual Meeting of the ISSS-2011, Hull, UK*. Vol. 55. 1. 2011.

[30] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. "Riskranker: scalable and accurate zero-day android malware detection". In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM. 2012, pp. 281–294.

[31] W. Enck, M. Ongtang, and P. McDaniel. "Understanding android security". In: *IEEE security & privacy* 7.1 (2009), pp. 50–57.

[32] W. Enck. "Defending users against smartphone apps: Techniques and future directions". In: *Proceedings of International Conference on Information Systems Security*. Springer. 2011, pp. 49–70.

[33] T.-E. Wei, A. B. Jeng, H.-M. Lee, C.-H. Chen, and C.-W. Tien. "Android privacy". In: *2012 International Conference on Machine Learning and Cybernetics*. Vol. 5. IEEE. 2012, pp. 1830–1837.

[34] A. Calvagna and E. Tramontana. "Delivering Dependable Reusable Components by Expressing and Enforcing Design Decisions". In: *Proc. of IEEE Computer Software and Applications Conf. (COMPSAC)*. 2013.

[35] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. "Android permissions demystified". In: *Proceedings of the 18th ACM conference on Computer and communications security*. ACM. 2011, pp. 627–638.

[36] Z. Aung and W. Zaw. "Permission-based android malware detection". In: *International Journal of Scientific & Technology Research* 2.3 (2013), pp. 228–234.

[37] S. M. Kywe, Y. Li, K. Petal, and M. Grace. "Attacking android smartphone systems without permissions". In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE. 2016, pp. 147–156.

[38] E. Tramontana and G. Verga. "Mitigating Privacy-related Risks for Android Users". In: *Proceedings of IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. 2019.

[39] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov. "Android permissions remystified: A field study on contextual integrity". In: *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 2015, pp. 499–514.

[40] E. Fernandes, J. Jung, and A. Prakash. "Security analysis of emerging smart home applications". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 636–654.

[41] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps". In: *Acm Sigplan Notices* 49.6 (2014), pp. 259–269.

[42] G. Ascia, V. Catania, R. Di Natale, A. Fornaia, M. Mongiovì, S. Monteleone, G. Pappalardo, and E. Tramontana. "Making android apps data-leak-safe by data flow analysis and code injection". In: *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE. 2016, pp. 205–210.

[43] J. P. Achara, M. Cunche, V. Roca, and A. Francillon. "Short paper: WifiLeaks: underestimated privacy implications of the access_wifi_state android permission". In: *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM. 2014, pp. 231–236.

[44] L. Nguyen, Y. Tian, S. Cho, W. Kwak, S. Parab, Y. Kim, P. Tague, and J. Zhang. "Unlocin: Unauthorized location inference on smartphones without being caught". In: *2013 International Conference on Privacy and Security in Mobile Systems (PRISMS)*. IEEE. 2013, pp. 1–8.

[45] J. Canny. "Collaborative filtering with privacy". In: *Proceedings IEEE Symposium on Security and Privacy*. 2002, pp. 45–57.

[46] J. Canny. "Collaborative filtering with privacy via factor analysis". In: *Proceedings of Annual international ACM SIGIR conference on Research and development in information retrieval*. 2002, pp. 238–245.

[47] H. Polat and W. Du. "SVD-based collaborative filtering with privacy". In: *Proceedings of ACM symposium on Applied computing*. 2005, pp. 791–795.

[48] R. Agrawal and R. Srikant. "Privacy-preserving data mining". In: *ACM Sigmod Record*. Vol. 29. 2. ACM. 2000, pp. 439–450.

[49] M. K. Reiter and A. D. Rubin. "Crowds: Anonymity for web transactions". In: *ACM transactions on information and system security (TISSEC)* 1.1 (1998), pp. 66–92.

[50] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. "Anonymous connections and onion routing". In: *Proceedings of IEEE Symposium on Security and Privacy*. 1997, pp. 44–54.

[51] B. Rashidi and C. J. Fung. "A Survey of Android Security Threats and Defenses." In: *JoWUA* 6.3 (2015), pp. 3–35.

[52] K. Bauer, H. Gonzales, and D. McCoy. "Mitigating evil twin attacks in 802.11". In: *2008 IEEE International Performance, Computing and Communications Conference*. IEEE. 2008, pp. 513–516.

[53] J. Lee, C. Tu, and S. Jung. "Man-in-the-middle attacks detection scheme on smartphone using 3g network". In: *The Fourth International Conference on Evolving Internet*. 2012, pp. 65–70.

[54] L. M. Adleman. "An abstract theory of computer viruses". In: *Conference on the Theory and Application of Cryptography*. Springer. 1988, pp. 354–374.

[55] F. Cohen. “Computational aspects of computer viruses”. In: *Computers & Security* 8.4 (1989), pp. 297–298.

[56] A. Mylonas, S. Dritsas, B. Tsoumas, and D. Gritzalis. “Smartphone security evaluation the malware attack case”. In: *Proceedings of the International Conference on Security and Cryptography*. IEEE. 2011, pp. 25–36.

[57] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan. “Android security: a survey of issues, malware penetration, and defenses”. In: *IEEE communications surveys & tutorials* 17.2 (2014), pp. 998–1022.

[58] T. Vidas, D. Votipka, and N. Christin. “All Your Droid Are Belong to Us: A Survey of Current Android Attacks.” In: *Woot*. 2011, pp. 81–90.

[59] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. “Privilege escalation attacks on android”. In: *international conference on Information security*. Springer. 2010, pp. 346–360.

[60] Y. Zhou and X. Jiang. “Dissecting android malware: Characterization and evolution”. In: *2012 IEEE symposium on security and privacy*. IEEE. 2012, pp. 95–109.

[61] L. Dua and D. Bansal. “Taxonomy: Mobile malware Threats and detection techniques”. In: *Computer Science & Information Technology (CS & IT)* (2014), pp. 213–221.

[62] K. Cheng, M. Gao, and R. Guo. “Analysis and research on HTTPS hijacking attacks”. In: *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*. Vol. 2. IEEE. 2010, pp. 223–226.

[63] L. Demir. “Wi-Fi tracking: what about privacy”. In: (2013).

[64] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. “Why Eve and Mallory love Android: An analysis of Android SSL (in) security”. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 50–61.

[65] M. Conti, N. Dragoni, and V. Lesyk. “A survey of man in the middle attacks”. In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 2027–2051.

[66] M.-W. Park, Y.-H. Choi, J.-H. Eom, and T.-M. Chung. “Dangerous Wi-Fi access point: attacks to benign smartphone applications”. In: *Personal and ubiquitous computing* 18.6 (2014), pp. 1373–1386.

[67] E. Dondyk and C. C. Zou. "Denial of convenience attack to smartphones using a fake Wi-Fi access point". In: *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. IEEE. 2013, pp. 164–170.

[68] H. Mustafa and W. Xu. "Cetad: Detecting evil twin access point attacks in wireless hotspots". In: *2014 IEEE Conference on Communications and Network Security*. IEEE. 2014, pp. 238–246.

[69] M. Chen, S. Mao, and Y. Liu. "Big data: A survey". In: *Mobile networks and applications* 19.2 (2014), pp. 171–209.

[70] V. N. Gudivada, R. Baeza-Yates, and V. V. Raghavan. "Big data: Promises and problems". In: *Computer* 3 (2015), pp. 20–23.

[71] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[72] J.-R. Hwang, H.-Y. Kang, and K.-J. Li. "Spatio-temporal similarity analysis between trajectories on road networks". In: *Proc. of Conf. on Conceptual Modeling*. Springer. 2005.

[73] M. Vlachos, D. Gunopulos, and G. Das. "Rotation invariant distance measures for trajectories". In: *Proc. of ACM Conf. on Knowledge Discovery and Data Mining*. 2004.

[74] W. Mathew, R. Raposo, and B. Martins. "Predicting future locations with hidden Markov models". In: *Proc. of ACM Ubiquitous Computing*. 2012.

[75] B Borah and D. Bhattacharyya. "An improved sampling-based DBSCAN for large spatial databases". In: *Intelligent Sensing and Information Processing, 2004. Proceedings of International Conference on*. IEEE. 2004, pp. 92–96.

[76] Z. Wang, M. Lu, X. Yuan, J. Zhang, and H. Van De Wetering. "Visual traffic jam analysis based on trajectory data". In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2159–2168.

[77] P. S. Castro, D. Zhang, and S. Li. "Urban traffic modelling and prediction using large scale taxi GPS traces". In: *Proc. of Pervasive Computing*. Springer. 2012.

[78] T. Biehle and H. Schultz. *Device, method, and computer program for providing traffic jam information via a vehicle-to-vehicle interface*. US Patent 10,380,885. 2019.

[79] J.-G. Lee, J. Han, and X. Li. "A unifying framework of mining trajectory patterns of various temporal tightness". In: *IEEE Transactions on Knowledge and Data Engineering* 27.6 (2015), pp. 1478–1490.

[80] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. "Recommender systems survey". In: *Knowledge-based systems* 46 (2013), pp. 109–132.

[81] P. Lops, M. De Gemmis, and G. Semeraro. "Content-based recommender systems: State of the art and trends". In: *Recommender systems handbook*. Springer, 2011, pp. 73–105.

[82] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. "Collaborative filtering recommender systems". In: *The adaptive web*. Springer, 2007, pp. 291–324.

[83] M. Aliannejadi and F. Crestani. "Personalized context-aware point of interest recommendation". In: *ACM Transactions on Information Systems (TOIS)* 36.4 (2018), p. 45.

[84] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann. "Time-aware point-of-interest recommendation". In: *Proceedings of the international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2013, pp. 363–372.

[85] B. Liu, Y. Fu, Z. Yao, and H. Xiong. "Learning geographical preferences for point-of-interest recommendation". In: *Proceedings of ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 1043–1051.

[86] H.-P. Hsieh, C.-T. Li, and S.-D. Lin. "Exploiting large-scale check-in data to recommend time-sensitive routes". In: *Proceedings of the ACM SIGKDD International Workshop on Urban Computing*. ACM. 2012, pp. 55–62.

[87] H. Yoon, Y. Zheng, X. Xie, and W. Woo. "Smart itinerary recommendation based on user-generated GPS trajectories". In: *International Conference on Ubiquitous Intelligence and Computing*. Springer. 2010, pp. 19–34.

[88] Y.-L. Hsueh and H.-M. Huang. "Personalized itinerary recommendation with time constraints using GPS datasets". In: *Knowledge and Information Systems* 60.1 (2019), pp. 523–544.

[89] D. T. Wagner, A. Rice, and A. R. Beresford. "Device analyzer: Understanding smartphone usage". In: *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer. 2013, pp. 195–208.

[90] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. "Understanding mobility based on GPS data". In: *Proceedings of the 10th international conference on Ubiquitous computing*. ACM. 2008, pp. 312–321.

[91] Y. Zheng, X. Xie, W.-Y. Ma, et al. "Geolife: A collaborative social networking service among user, location and trajectory." In: *IEEE Data Eng. Bull.* 33.2 (2010), pp. 32–39.

[92] Y. Zheng, L. Wang, R. Zhang, X. Xie, and W.-Y. Ma. "GeoLife: Managing and understanding your past life over maps". In: *Mobile Data Management, 2008. MDM'08. 9th International Conference on*. IEEE. 2008, pp. 211–212.

[93] M. Kiermeier and M. Werner. "Similarity search for spatial trajectories using online lower bounding DTW and presorting strategies". In: *Leibniz International Proceedings in Informatics, LIPIcs 90 (2017)* 90 (2017), p. 18.

[94] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. "T-drive: driving directions based on taxi trajectories". In: *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. ACM. 2010, pp. 99–108.

[95] J. Yuan, Y. Zheng, X. Xie, and G. Sun. "Driving with knowledge from the physical world". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 316–324.

[96] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi. *CRAWDAD dataset roma/taxi (v. 2014-07-17)*. Downloaded from `https://crawdad.org/roma/taxi/20140717`. July 2014. DOI: `10.15783/C7QC7M`.

[97] E. Wang, Y. Yang, J. Wu, W. Liu, and X. Wang. "An efficient prediction-based user recruitment for mobile crowdsensing". In: *IEEE Transactions on Mobile Computing* 17.1 (2017), pp. 16–28.

[98] T. Murakami and H. Watanabe. "Localization attacks using matrix and tensor factorization". In: *IEEE Transactions on Information Forensics and Security* 11.8 (2016), pp. 1647–1660.

[99] M. Kurmis, M. Voznak, G. Kucinskas, D. Drungilas, Z. Lukosius, S. Jakovlev, and A. Andziulis. "Development of method for service support management in vehicular communication networks". In: *Advances in Electrical and Electronic Engineering* 15.4 (2017), pp. 598–605.

[100] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. "One-pass error bounded trajectory simplification". In: *Proceedings of the VLDB Endowment* 10.7 (2017), pp. 841–852.

[101] Y. Xu, X. Wu, and Q. Wang. *Sentiment Analysis of Yelp's Ratings Based on Text Reviews*. 2015.

[102] A. Sadeghi, H. Bagheri, J. Garcia, and S. Malek. "A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software". In: *IEEE Transactions on Software Engineering* 43.6 (2017), pp. 492–530.

[103] M. Mongiovi, G. Giannone, A. Fornaia, G. Pappalardo, and E. Tramontana. "Combining static and dynamic data flow analysis: a hybrid approach for detecting data leaks in Java applications". In: *Proc. of Symposium on Applied Computing (SAC)*. ACM, 2015.

[104] C. Willems, T. Holz, and F. Freiling. "Toward automated dynamic malware analysis using cwsandbox". In: *IEEE Security & Privacy* 5.2 (2007).

[105] U. Bayer, C. Kruegel, and E. Kirda. "TTAnalyze: A tool for analyzing malware". In: *Proc. of Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*. 2006.

[106] R. Moskovitch, Y. Elovici, and L. Rokach. "Detection of unknown computer worms based on behavioral classification of the host". In: *Computational Statistics & Data Analysis* 52.9 (2008), pp. 4544–4566.

[107] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer. "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey". In: *Information security technical report* 14.1 (2009), pp. 16–29.

[108] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici. "Improving malware detection by applying multi-inducer ensemble". In: *Computational Statistics & Data Analysis* 53.4 (2009), pp. 1483–1494.

[109] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov. "Dynamically Regulating Mobile Application Permissions". In: *IEEE Security & Privacy* 16.1 (2018), pp. 64–71.

[110] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens. "Drebin: Effective and explainable detection of android malware in your pocket." In: *Proc. of Network and Distributed System Security (NDSS) Symposium*. Vol. 14. 2014, pp. 23–26.

[111] C.-I. Fan, H.-W. Hsiao, C.-H. Chou, and Y.-F. Tseng. "Malware detection systems based on API log data mining". In: *Proc. of computer software and applications conference*. Vol. 3. IEEE. 2015, pp. 255–260.

[112] A. Distefano, A. Fornaia, E. Tramontana, and G. Verga. "Detecting Android Malware According to Observations on User Activities". In: *Proc. of International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE. 2018, pp. 241–246.

[113] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot. "A conceptual view on trajectories". In: *Data & Knowledge Engineering* 65.1 (2008), pp. 126–146.

[114] Y. Cui and S. S. Ge. "Autonomous vehicle positioning with GPS in urban canyon environments". In: *IEEE Trans. Rob. Autom.* 19.1 (2003), pp. 15–25.

[115] F. Banno, D. Marletta, G. Pappalardo, and E. Tramontana. "Tackling consistency issues for runtime updating distributed systems". In: *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. IEEE. 2010, pp. 1–8.

[116] R. Pérez-Torres, C. Torres-Huitzil, and H. Galeana-Zapién. "Full on-device stay points detection in smartphones for location-based mobile applications". In: *Sensors* 16.10 (2016), p. 1693.

[117] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of International Conference on Knowledge Discovery and Data Mining*. 1996, pp. 226–231.

[118] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. "IP geolocation databases: Unreliable?" In: *ACM SIGCOMM Computer Communication Review* 41.2 (2011), pp. 53–56.

[119] J. Cruz, E. Silva, R. J. Rossetti, D. C. Silva, E. C. Oliveira, and J. Neto. "Application of multi-agent systems to shared transport services: A Review". In: *Proceedings of Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE. 2018, pp. 1–6.

[120] Q. Ye, Z. Zhang, and R. Law. "Sentiment classification of online reviews to travel destinations by supervised machine learning approaches". In: *Expert Systems with Applications* 36.3 (2009), pp. 6527–6535.

[121] S. Akter and M. T. Aziz. "Sentiment analysis on facebook group using lexicon based approach". In: *2016 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*. IEEE. 2016, pp. 1–4.

[122] N. M. Ali, A. El Hamid, M. Mostafa, and A. Youssif. "Sentiment Analysis for Movies Reviews Dataset Using Deep Learning Models". In: *Aliaa, Sentiment Analysis for Movies Reviews Dataset Using Deep Learning Models (June 14, 2019)* (2019).

[123] P. Palanisamy, V. Yadav, and H. Elchuri. "Serendio: Simple and Practical lexicon based approach to Sentiment Analysis". In: *Proceedings of Second Joint Conference on Lexical and Computational Semantics*. 2013, pp. 543–548.

[124] S. Tan and J. Zhang. "An empirical study of sentiment analysis for chinese documents". In: *Expert Systems with applications* 34.4 (2008), pp. 2622–2629.

[125] A. E. Fano. "Shopper's eye: using location-based filtering for a shopping agent in the physical world". In: *Agents*. Vol. 98. 1998, pp. 416–421.

[126] P. Xuan and V. Lesser. "Multi-agent policies: from centralized ones to decentralized ones". In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*. ACM. 2002, pp. 1098–1105.

[127] C. Boutilier. "Sequential optimality and coordination in multiagent systems". In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 99. 1999, pp. 478–485.

[128] P. Xuan, V. Lesser, and S. Zilberstein. "Communication decisions in multi-agent cooperation: Model and experiments". In: *Proceedings of the fifth international conference on Autonomous agents*. ACM. 2001, pp. 616–623.

[129] J. Z. Hernández, S. Ossowski, and A. Garcıa-Serrano. "Multiagent architectures for intelligent traffic management systems". In: *Transportation Research Part C: Emerging Technologies* 10.5-6 (2002), pp. 473–506.

[130] R. Giunta, G. Pappalardo, and E. Tramontana. "Aspects and Annotations for Controlling the Roles Application Classes Play for Design Patterns". In: *Proc. of IEEE Asia Pacific Software Engineering Conference (APSEC)*. Ho Chi Minh, Vietnam, 2011, pp. 306–314.

[131] D. M. Jiménez-Bravo, J. Pérez-Marcos, D. H. De la Iglesia, G. Villarrubia González, and J. F. De Paz. "Multi-Agent Recommendation System for Electrical Energy Optimization and Cost Saving in Smart Homes". In: *Energies* 12.7 (2019), p. 1317.

[132] A. Casali, L. Godo, and C. Sierra. "A tourism recommender agent: from theory to practice". In: *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial* 12.40 (2008), pp. 23–38.

[133] A. Casali, L. Godo, and C. Sierra. "Validation and Experimentation of a Tourism Recommender Agent based on a Graded BDI Model." In: *CCIA*. 2008, pp. 41–50.

[134] M. Batet, A. Moreno, D. Sánchez, D. Isern, and A. Valls. "Turist@: Agent-based personalised recommendation of tourist activities". In: *Expert Systems with Applications* 39.8 (2012), pp. 7319–7329.

[135] A. B. Othmane, A. Tettamanzi, S. Villata, N. Le Thanh, and M. Buffa. "A Multi-context Framework for Modeling an Agent-based Recommender System." In: *ICAART (2)*. 2016, pp. 31–41.

[136] C.-C. Chen and J.-L. Tsai. "Determinants of behavioral intention to use the Personalized Location-based Mobile Tourism Application: An empirical study by integrating TAM with ISSM". In: *Future Generation Computer Systems* (2017).

[137] L. Sebastia, A. Giret, and I. Garcia. "A multi agent architecture for tourism recommendation". In: *Trends in Practical Applications of Agents and Multiagent Systems*. Springer, 2010, pp. 547–554.

[138] L. Sebastiá Tarín, A. S. Giret Boggino, and I. García García. "A multi agent architecture for single user and group recommendation in the tourism domain". In: *International Journal of Artificial Intelligence* 6.11 (2011), pp. 161–182.

[139] B. Pang and L. Lee. "Opinion mining and sentiment analysis". In: *Foundations and Trends in Information Retrieval* 2.1–2 (2008), pp. 1–135.

[140] A. Abbasi, H. Chen, and A. Salem. "Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums". In: *ACM Transactions on Information Systems (TOIS)* 26.3 (2008), p. 12.

[141] E. Kouloumpis, T. Wilson, and J. Moore. "Twitter sentiment analysis: The good the bad and the omg!" In: *Fifth International AAAI conference on weblogs and social media*. 2011.

[142] V. Vyas and V Uma. "Approaches to sentiment analysis on product reviews". In: *Sentiment Analysis and Knowledge Discovery in Contemporary Business*. IGI Global, 2019, pp. 15–30.

[143] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede. "Lexicon-based methods for sentiment analysis". In: *Computational linguistics* 37.2 (2011), pp. 267–307.

[144] N. Jindal and B. Liu. "Identifying comparative sentences in text documents". In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2006, pp. 244–251.

[145] B. Liu. "Sentiment analysis and subjectivity". In: *Handbook of Natural Language Processing, Second Edition*. Chapman and Hall/CRC, 2010, pp. 627–666.

[146] R. Giunta, G. Pappalardo, and E. Tramontana. "AODP: refactoring code to provide advanced aspect-oriented modularization of design patterns". In: *Proceedings of ACM Symposium on Applied Computing (SAC)*. 2012.

[147] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. "Aspect-oriented programming". In: *European conference on object-oriented programming*. Springer. 1997, pp. 220–242.

[148] R. Laddad. *Aspectj in action: enterprise AOP with spring applications*. Manning Publications Co., 2009.

[149] A. Sinkov and T. Feil. *Elementary cryptanalysis*. Vol. 22. Maa, 2009.