Federal University of Amazonas
Institute of Computing
Graduate Program in Computer Science

Functionality-Based Mobile Application Recommendation System with Security and Privacy Awareness

THIAGO DE SOUZA ROCHA

Functionality-Based Mobile Application Recommendation System with Security and Privacy Awareness

Thesis submitted to the Graduate Program of the Institute of Computing of the Federal University of Amazonas to obtain the title of Doctor in Informatics

ADVISOR: PhD. EDUARDO JAMES PEREIRA SOUTO

Manaus-AM

February 2020

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

R672m

Rocha, Thiago de Souza
Functionality-Based Mobile Application Recommendation System with Security and Privacy Awareness / Thiago de Souza Rocha . 2020
84 f.: il. color; 31 cm.

Orientador: Eduardo James Pereira Souto
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Security. 2. Privacy. 3. Recommendation. 4. Malware. I. Souto, Eduardo James Pereira. II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

Functionality-Based Mobile Application Recommendation System with Security and Privacy Awareness"

THIAGO DE SOUZA ROCHA

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Eduardo James Pereira Souto - PRESIDENTE

Prof. Edleno Silva de Moura - MEMBRO INTERNO

Prof. Eduardo Luzeiro Feitosa - MEMBRO INTERNO

Prof. Djamel Fawzi Hadj Sadok - MEMBRO EXTERNO

Prof. André Ricardo Abed Grégio - MEMBRO EXTERNO

Manaus, 06 de Março de 2020

Acknowledgement

First to God, for being responsible for my existence and for giving me strength throughout my path.

To my parentes José Francisco and Mirlene Rocha and to my brother Arnaldo Rocha, for the support.

To my supervisor, Prof. Eduardo Souto, mainly for the patience, and for believe in me.

To professor Khalil El-Khatib from University of Ontario (UOIT) for contributing to the thesis.

To the teachers, friends and all the staff from ICOMP, who contributed to the conclusion of this work.

To FPF Tech for giving me structure and time for the development of the work.

To my grandfather Arnaldo Felisberto Imbiriba da Rocha, for being the greatest motivator of my master's and PhD degrees.

To my girlfriend Tatiane Ribeiro for supporting me all the time.

To all my other family members for always encouraging me.

And to all my friends! Thanks!

“Be the change you want to see in the world”

Abstract

Nowadays, with the advent of mobile devices, there are a variety of mobile applications to execute daily tasks, such as paying bills, watching movies and ordering food. That popularity caught the attention of malicious developers that started creating malicious applications for mobile devices instead of desktop computers. Some malicious applications claim that they can perform a certain common task, such as paying bills, just to lure users to install the application to damage their devices and/or execute malicious activities such as sending premium SMS messages or leaking users personal sensitive information. Because of that, users need a way to choose an app that is considered safe and meets their needs. For instance, if a user wants to change the application that he uses to order food, the list of suggestions must have only applications classified as benign that are capable of ordering food. Recommendation systems are currently being used to choose applications inside the Android environment, but most approaches do not evaluate security and privacy, and when they do, only the applications permissions configuration are considered. However, recent studies demonstrate that this approach is not enough. In addition, some approaches rely on user's knowledge about the permissions, which studies have also shown that is error prone because most of users do not understand how the permissions system work. In this context, this work presents a novel functionality-based recommendation system with security and privacy awareness to evaluate and suggest apps. The system consists of a machine learning security layer that evaluates the applications to make sure that only apps classified as benign can be suggested. The proposed system also has an application scoring system that is based on functionality to ensure that only the applications with similar purposes can be suggested. In addition, users will be able to see popularity, usability and privacy metrics and add weights so that suggestions are made according to the user's preferences. Furthermore, a mapping between the permissions, application method calls, and descriptions is made to create phrases so that users can understand what the application being evaluated can do on the mobile device. The goal is to provide comprehensible information so users will be able to check if the application is executing any suspicious behavior and/or if it is requesting too much permissions. A prototype was developed and compared with works from the literature and the experiments demonstrated that the system had better results because it was able to suggest only applications classified as benign that have similar behaviors. The prototype was also compared with the official Google Play Store in order to verify if the list of suggestion has only apps with similar goals. The results demonstrate that, in terms of functionality, the prototype suggestion list had only apps that share similar goals and that Google Play categories needs to be better defined. The main contributions are the recommendation system with the advent of a security layer, the app scoring system inside a functionality context and the mapping between permissions and API calls raising user confidence and understanding.

Keywords: Security, Privacy, Recommendation, Malware, Android.

Figures List

Tables List

Sumário

Chapter 1

Introduction

The number and usage of mobile devices has increased dramatically over the last decade and has changed the way users execute their daily tasks and do business and will continue to do so. The number of global smartphone users has already exceeded 2 billions and is expected to reach 3 billions by 2020 [1]. As a result, the number of applications developed to help users perform a large number of tasks has also grown considerably. In 2018, the official Android store (Google Play) contained 2.1 billion apps [2] and the number of downloads was forecasted to reach 258.2 billion by 2020 [3]. Such applications are used daily and provide various functionalities such as phone calls, e-mail sending, GPS service, and camera to list a few.

Due to the large number of apps, recommendation systems have been used by users to find applications that suit their needs and interests. For example, Google Play recommends applications based on aspects of the app being searched, such as store category and the name of the app developer(s). However, security and privacy aspects are not satisfactorily considered by the recommendation systems, both in official stores and in recommendation systems in general [4].This is a concern since applications frequently request users sensitive or private data such as logins, passwords, location and financial information to accomplish their objectives. Therefore, these applications became potential targets for malicious developers.

According to IDC [5], Google Android platform owns 86% of the global smartphone market and, therefore, is the preferred target for malicious applications developers. The Symantec Internet Security Threat Report [6] shows that one in every five Android applications (or apps) were actually malicious. Since Play Store was released malicious applications have been discovered within the store. Recently, Google removed 17,000 Android apps that were carrying the Joker malware [7]. These malicious Android applications were sending fraudulent text messages and charging people for fake services or were using Wireless Application Protocol (WAP) billing fraud. Other examples can be found in [8], [9], [10].

Even the applications that come preinstalled inside mobile devices can be malicious and pose risks. For instance, security experts found a weather application that is preinstalled on Alcatel smartphones that was subscribing device owners phone numbers to premium SMS services without user consent and was also accessing private user information such as phone IMEI, location, email address, which it was sending to a third-party server [11]. As an attempt to minimize these problems, Google released Google Play Protect in 2017 [12], an application that scans apps automatically, even after installation to ensure they remain safe. However, several issues regarding Play Protect have been reported.

First, sometimes it blocks legitimate developer apps and/or allows malwares [13], [14]. Second, it sometimes blocks private business apps and device manufactures apps that are not installed from the official store [15]. Third, it keeps running from time to time, consuming the device battery. Fourth, Google Play Protect was ranked as the worst antivirus of twenty mobile antivirus applications in an AV-Test software evaluation test in May 2019 [16]. Finally, Play Protect does not recommend an alternative application if it flags an app as potentially malicious. Mobile users need an alternative app that is safe and provides similar functionalities to those which the user desires from the app being evaluated. All these examples show the need for a system that is able to evaluate mobile applications with security and privacy awareness and also can suggest other apps with similar functionalities if the app being evaluated is malicious or has any suspicious behavior.

There are some recent studies about recommendation systems that consider some security and privacy aspects. However, these works also have several problems and limitations. First, most of them only check app permissions configuration. Permission use is an important feature for calculating application security risks. However, they are not sufficient to guarantee that an application is safe [17],[18]. Some papers and security forums have already demonstrated attacks that can happen without any Android permission usage [19],[20]. Besides that, most users cannot understand how permissions work, what they do or do not pay attention when permissions are requested [21]. Such fact creates a gap between user expectations and application behavior [22]. Moreover, there is also a problem with over permission [4],[23] when an app requests more permissions than necessary. This situation can lead to the appearance of attacks from permissions that are not even necessary by the apps.

Another limitation is related to the way application recommendations are made. Most of the time, these recommendations are based on a set of applications that belong to the same category as the official store. This may not satisfy the needs of a user who is looking for a specific functionality. For instance, when a user is searching for an application that is similar to WhatsApp, if the recommendation system returns Facebook application, as it happens in the Play Store, it would be unsatisfactory, although both belong to the same category. In mobile app recommendation, metrics should be calculated inside a functionality context and not by category. For instance, if a user wants to change the app

that is used to order food, the recommendation system should consider only safe applications that can also order food.

Privacy must also be calculated inside a functionality context to detect the leak of sensitive information. In some cases, the permissions and API calls to access sensitive information that may be considered malicious in one app could be a feature in another app [24]. For instance, calculating the privacy score using location permissions and API calls from an app that tracks current user location may be considered if it is an app with bank functionalities but should not be considered if it is a navigation app, because in this case the request is benign and the location information is necessary for the correct execution of tis service.. Moreover, most users are not aware of the data collected by apps [25].

To overcome these problems, we propose a system to evaluate and recommend mobile applications, inside Android operational system environment, with security and privacy awareness. The proposed system has a security layer that evaluates an application and classifies it as being malign or benign. Thus, only applications classified as benign (safe) are considered in the recommendation phase. Also, we employ a technique, called Logical Predicate Mapping (LPM), which allows users to understand the permissions and API calls requested by the app, as well as privacy risks. This way users can decide what to do and understand what can happen.

1.1. Objectives

This thesis provides a functionality-based mobile application recommendation system with security and privacy awareness. The first challenge is the capability to create a method that can evaluate a target application and classify it as being malign or benign before a recommendation is made. Second, the possibility to apply topic extraction techniques on applications descriptions to suggest only apps with the same functionality. Moreover, the system analyzes the information gathered from the target application and calculates metrics such as privacy, usability, popularity and checks the permissions and API calls to map all possible behaviour that could cause privacy and security risks or any behavior that is not aligned with the application description. Finally, the system presents a summary to the users that groups up all the risk and privacy related information. This way, users can decide what to do and understand what could possibly happen in their mobile devices. To reach these goals this thesis presents the following specific objectives:

1. Creation of a method to download apps from an application store;
2. Creation of a mechanism that extracts the features from the applications to create and train a model that classifies applications into benign or malign;
3. Development of a functionality-based recommendation engine that is able to suggest apps that share similar goals;

4. Development of functionality-based algorithms that can calculate usability, privacy and popularity metrics to build a ranking of applications to be suggested to the users;
5. Creation of an understandable summary with the information gathered during the evaluation in a way that it is possible for technical and non-technical users to understand.

1.2. Contributions

From the objectives defined in the previous section, this work offers the following contributions:

- A mobile application recommendation system with a machine learning security layer that evaluates apps and only suggests the ones classified as benign;
- A functionality-based app scoring system that was created to obtain functionality, privacy, usability and popularity metrics to later rank the apps that are suggested to the users. Since the scoring system is based on the purpose of the apps, all the metrics are calculated inside a functionality context. With that it is possible to only recommend applications that perform similar functionalities as the application being evaluated and also check the privacy, popularity and usability in different conditions. For instance, in relation to privacy, in some cases the permissions and API calls that may be considered malicious in one application could be a feature in another app [28];
- Creation of a novel Logical Predicate Mapping (LPM) that aims to clarify the behavior that a given application can execute inside mobile devices so mitigation actions can be taken and problems such as overpermission can be detected and faced.

The results obtained from this work originated the following scientific productions written so far:

- Rocha, T., El-Khatib, K. & Souto, E. (2019), “Techniques to Detect Data Leakage in Mobile Applications”, International Journal of Security and Networks (IJSN), february;
- Rocha, T., & Souto, E. (2019), "Avaliação e Recomendação de Aplicativos para Dispositivos Móveis com Foco em Segurança e Privacidade.", XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2019 - Artigos Completos / Full Papers), São Paulo (SP), september;
- Rocha, T., El-Khatib, K. & Souto, E. (2020), “Functionality-Based Mobile Application Recommendation System with Security and Privacy Awareness”, Computers & Security (COSE). (Under review).

In addition to these publications, the author of this research had the opportunity to contribute to other publications using the knowledge obtained during the execution of this work:

- Sousa, W., Souto, E., Rocha, T., Pazzi, R., Pramudianto, F. (2015) “User activity recognition for energy saving in smart home environment”, IEEE Symposium on Computers and Communication (ISCC), Larnaca (CY), july;
- Rocha, T., & Souto, E. (2016), “Automatização de teste de software com ferramentas de software livre”, Elsevier Editora LTDA (Book Chapter);
- Sousa, W., Souto, E., Rocha, T., Pereira, N., Kool, P., Rosengren, P. (2016), “Prediction of electrical energy consumption for Internet of Things in disaggregated databases”, IEEE Symposium on Computers and Communication (ISCC), Messina (IT), june;
- Rocha, T., Monteiro, A., Cassio, B., Minatel, P., Silva, B., Boeira, F., Azulay, D., & Souto, E. (2016), “Data leakage detection in Tizen Web applications”, Annual Conference on Privacy, Security and Trust (PST), Auckland (NZ), December;
- Rocha, T., Damasceno, A., & Souto, E. (2018), " TaintJSec: Um Método de Análise Estática de Marcação em Código Javascript para Detecção de Vazamento de Dados Sensíveis.", XVIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2018 - Artigos Completos / Full Papers), Natal (RN), october.

1.3. Document Structure

The rest of this thesis is organized as follows:

Chapter 2 presents the main concepts related to the understanding of this work. Initially Android is explained with details on its framework, architecture, main components, applications and security and privacy tips for developers. Next, the techniques and algorithms that are used to build the system such as Taint Analysis, Machine Learning and Probabilistic Topic Models are explained by informing how they work, their types, strengths and weaknesses and how they are used in the proposed system.

Chapter 3 presents some works that were developed to recommend applications to users in Android environment and also evaluates security and privacy aspects. Each application is explained in terms of objectives, process, strengths and limitations. To better describe the approaches a table is shown at the end of the chapter showing some information from each work such as the features used and the technique employed. Finally, a discussion is made about the works focusing on the limitations and looking for improvements.

Chapter 4 presents the proposed system showing its architecture and explaining with details on how each phase works. Chapter 5 explains the details of the experiments carried out, depicts the dataset and describes the several test cases were created to evaluate the system comparing it with the most recent works and also with the official Google Play Store. Finally, Chapter 6 provides the final conclusions and directions for future work.

Chapter 2

Background

This chapter explains the basic concepts for the understanding of this thesis. It starts with an explanation about Android with its architecture and its applications security model. After, it presents taint analysis, that is one of the main techniques that are used to extract features from applications and finishes with an explanation about machine learning and probabilistic topic models.

2.1. Android OS and Application Structure

The architecture of the Android OS and its application. Android is an open source operating system maintained by Google, and is today the main open source platform for mobile devices, its source code can be found in [29] with information and instructions needed to create custom variants of Android and also to port acessories and devices to the Android platform.

2.1.1. Android Framework Architecture

Android platform is a stack of components, which is divided into five layers and six main sections as shown in Figure 2-1. It includes the operating system, a middleware, applications frameworks and some applications that provide basic functions to mobile devices [30].

Starting from the bottom, is a Linux Kernel section customized for embedded environments that has limited resources and its goal is to cover system functionality such as low-level memory management, power management, alarm, binder to handle Inter Process Communication (IPC) and runtime environment. Besides that, Linux Kernel deals with device drivers and networking. Using a Linux Kernel also allows Android to use Linux key security features.

Second layer contains the Hardware Abstraction Layer (HAL) section and consists of a set of libraries that provides an interface for a specific type of hardware component such as Bluetooth and Audio. When a Framework API tries to access any device hardware the

HAL is responsible to load the hardware modules. Thus, HAL works as a communication layer between the Kernel and the upper layers.

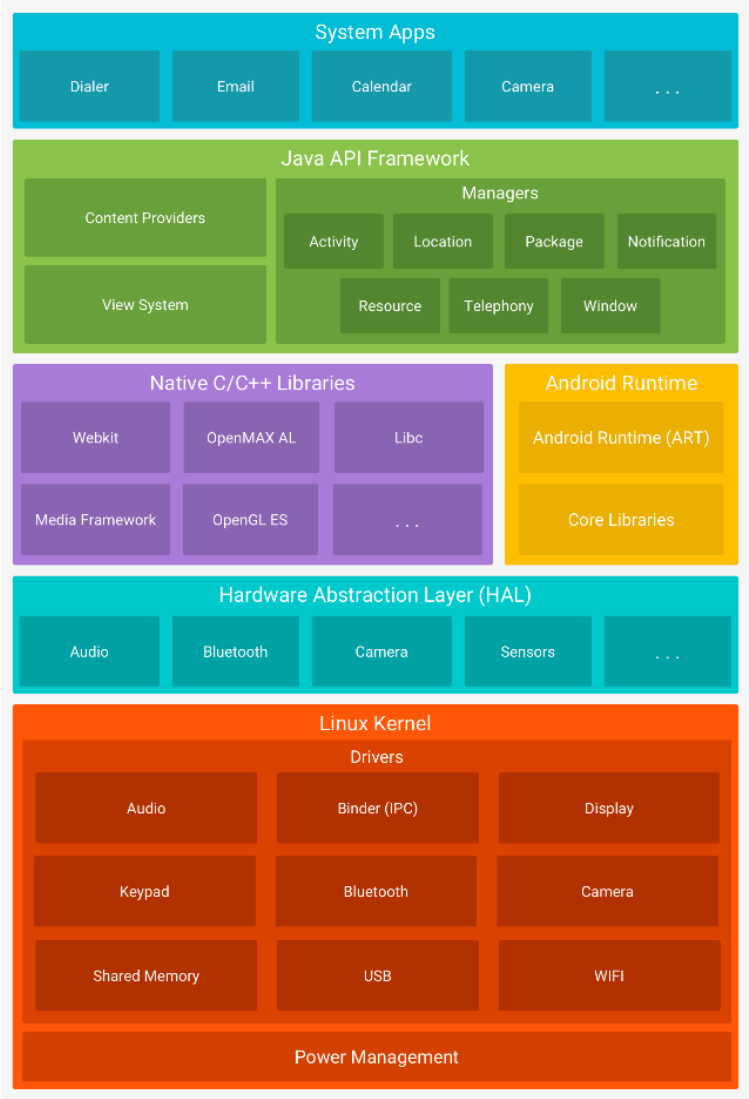

Figure 2-1: Android Framework Architecture.

The Third layer is composed by the Android runtime, core Java libraries and some Native C/C++ libraries. Devices with Android version above 5.0 use ART runtime while devices with a version below 5.0 use Dalvik, apps that work well on ART should also work on Dalvik but the reverse may be a problem. ART is an application runtime environment created to execute multiple virtual machines using limited device memory and executing .dex files, a more compressed bytecode format designed only for the Android platform that is optimized for minimal memory footprint. ART also has some other features such as optimized garbage collection, better debugging support and executes each application with its own process so the app cannot interfere with the operational system or other applications.

The Core Java libraries are used to provide functionalities about Java programming language so that Android applications developers can write programs for mobile devices using Java. Many Android platform components, such as the runtime, are built from native code using libraries written in C and C++. The Android platform provides Java framework APIs to make available the functionality of these native libraries to apps. For instance, OpenGL can be accessed through Java OpenGL API to manipulate graphics in an app using 2D and 3D objects. In addition, if someone is developing an app that requires C or C++ code, Android Native Development Kit (NDK) can be used to access these native platform libraries directly from the application native code.

In fourth layer the Java API Framework is composed of a set of services and Application programming interfaces (APIs) as Java classes that provide abstractions to access the underlying native libraries and runtime capabilities to applications, these services are available for developers to create their own Android applications. These APIs and services were created to implement the concept that Android applications are constructed from interchangeable, reusable and replaceable components, through this concept application developers have access to the same framework APIs that Android system apps use. Some key managers are:

- **Activity Manager:** Used to manage all the aspects of the applications including its lifecycle and a navigation back stack.
- **Content Provider:** Enables applications to access data from other apps, such as the contacts list and to share its own data between applications.
- **Notification Manager**: Used to display custom alerts and notifications to the users in the status bar. It is also used to notify users about services running in background.
- **View System:** An extensible and rich system used to build applications User Interface (UI), including buttons, lists, grids, text boxes, buttons, web browsers and other components.
- **Resource Manager:** Enables access to a set of non-code embedded resources such as strings, user interface apps layouts, graphics, figures, icons and color settings.
- **Telephony Manager**: Provides information about the telephony status, services available and handles settings of network connections.
- **Package Manager**: Used to provide information about the applications installed on device.
- **Location Manager**: Provides information about the location services.

The last layer is the system applications. By default Android comes with a couple of built-in applications that are used to execute basic services such as e-mail sending, SMS messaging, calendar, contacts, internet browsing and more. However, programmers may develop their own applications to execute these services and replace the Android system applications. Developers can also access these built-in apps from the app that they are

developing. For instance, if a developer would like an app to deliver an SMS message, its not necessary for him to develop the functionality, instead the developer could invoke the SMS built-in app that is already installed on the mobile device to deliver a message to the recipient.

2.1.2. Applications Security Mechanism

Following its architecture Android has some concepts and principles designed for application security. When talking about application security, development best practices have been created in an attempt to prevent or deny system exploitation, a security checklist can be found in [31] with the following tips:

1. **Store data safely:** Minimize the use of APIs to access sensitive data, do not expose sensitive data to third-party components if the goal of requesting this data is unclear. If data has to be accessed try to avoid sending it through the network or use a hash to protect it and verify external storage when it is used.
2. **Enforce secure communication:** Use Transport Layer Security (TLS) as an aditional layer for encrypted communication between clients and servers when transporting data through the network.
3. **Update security provider:** Use Google Play Services to automatically check and update the device security provider to avoid exploitation when vulnerabilities are found.
4. **Pay attention to permissions:** Permissions requests protect user sensitive data by limiting the applications access to information that is considered sensitive such as phone contacts and location information [32] and should be used only when the information is necessary for the application to execute or work. Android 6.0 Marshmallow, which was released in 2015, changed the way that application permissions are treated. With this new model, the users can add and revoke permissions at any time, which theoretically improves the user visibility and control over permissions but most of the users still do not understand the purpose of the permission they are being asked to allow or deny and that is why developers should pay attention and request permissions only when necessary. Besides that, the permissions requested from third-party libraries should also be verified because when a library is used it carries out all the permissions requested.

Android also has an application sandbox that executes each application separately as if it were a single user. Thus, each application has its own exclusive user id (UID) and a private and restricted directory to save data that are added during the installation process. The sandbox basically denies all requests that the app performs unless the permissions are granted. For instance, an app can share files by configuring its file mode to be world writable or readable. This model can be broken only by applications that have

the same digital certificate and that explicitly ask for the same UID at Android app configuration file.

Android components of an application such as activities, services and broadcasts can also be restricted to ensure that only certain allowed sources can execute a particular task, the permissions are added to the manifest file. For instance, an activity permission is added using the android:permission attribute inside the <activity> tag and checked during Context.startActivity() and Activity.startActivityForResult() methods, if the caller does not have the required permission a SecurityException is thrown. Content providers can also grant permissions and restrict who can have access to read and write data, more information about security in Android can be found in [33] and [34].

2.1.3. Application Structure

An Android Application is a zip format archive that contains several files and folders compressed into a package with the .apk (Android Package Kit) extension [35]. Figure 2-2 has an APK content example with the main files that make up an APK, each file is described below:

- **AndroidManifest.xml:** Android configuration file containing predefined elements such as application name, permissions, application components, libraries, icons and filters. The file is in binary format and can be converted into human-readable XML using tools such as apktool and Androguard.
- **classes.dex:** Application code compiled in the DEX format to be understandable by the Android runtime.
- **resources.arsc:** File with pre-compiled resources, such as binary XML files.
- **res/:** Folder that has the resources that are not pre-compiled such as layouts, strings, colors, drawables and icons. The R class contains the identifiers from all the resources.
- **META-INF/:** Folder that contains meta information to the signature of the application such as CERT.RSA, CERT.SF and Manifest.MF.
- **lib/:** Folder that contains compiled code specifically for a processor. The folder is divided into more folders to organize the code. For instance, x86 has compiled code for x86 processors only while armeabi-v7a has compiled code for ARMv7 and above based processors.

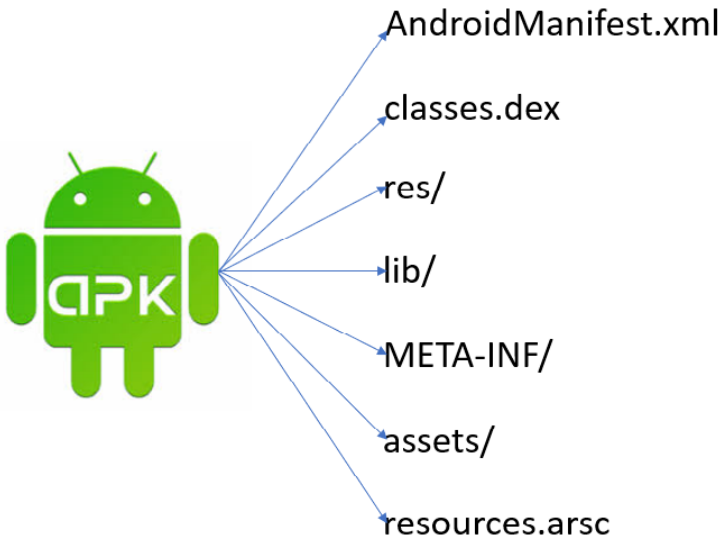

Figure 2-2: Android Application Package Kit Content.

2.1.4. Application Components

Android has four different types of application components [36]. Each one has different lifecycles and executes a different task with a specific purpose. An app does not necessarily need all these components, but to present a User Interface (UI) it has to contain at least an Activity, the components are listed below:

- **Activities:** Activities represents a single screen with a user interface in an Android application. They have its own lifecycle controlled by inherited methods and also groups up the visual elements (called views) of an app and allows users interaction with the screen. Each application is formed by multiple activities and the transition between them is made with Intents that is an asynchronous message that answers to system and applications requests and also binds multiple components.
- **Services:** Services are components with no graphical interface that run in background. They were created to execute tasks that take a large amount of time without disrupting the interaction of the user with the application. For instance, download a large amount of data such as musics, movies, or books from a webservice without compromising user experience.
- **Content Providers:** Content Providers are responsible for data storage and retrieval in Android applications. They can also be used to share data between multiple applications if the correct permissions are given. Android already has default content providers for some data such as contacts, images, videos and others. They can be accessed through an URI that determines the provider, a field name and the data type of the field.

- **Broadcast Receivers:** Just like a service, Broadcast Receivers are components that have no graphical interface and run in background. However, they are used for short term tasks. Broadcast Receivers are used to answer to intents that are sent from the operational system or from other applications. For instance, when an Android phone starts or the phone battery is low a broadcast is sent to all the applications in the phone and a broadcast receiver can be created to intercept this message and execute a task. Although Broadcast Receivers have no interface they can exhibit notifications in the status bar to alert users.

2.2. Information Flow and Taint Analysis

One way to verify applications security and privacy is to track sensitive data. In Android, Information Flow is the transfer of information from an information source such as accessing contacts list to an information sink such as sending a SMS message [37]. In a mobile environment, this is information being acquired by an application and being transferred to other app or to the internet, if this is done insecurely with sensitive data a data leakage may occur.

Taint analysis is the technique used to track information flow through a program analysis which adds labels or tags to a variable within the application under test. This variable is considered tainted while the rest are untainted. In general, taint analysis is defined as the process of tracing how tainted data propagates through a program as it executes [38]. In other words, if an information flows from an object y to an object x, denoted y -> x and y is tainted the analysis will have the ability to track the information. Taint Analysis has some concepts that are listed below:

- **Tag:** Act to add a label in each object that needs to be traced. For instance, user location.
- **Source:** It is where the application can get the sensitive information. It is the starting point to add the tags and create the tainted objects to be tracked. An example of a taint source is a function that reads data from an untrusted data source, for instance, the Android Location manager is the source to get information about the GPS.
- **Propagation:** Is the process of executing the propagation of the tag while the program is executed. For example, when an untainted variable in the code receives information about location coordinates, then its status changes to tainted and will be tracked during the execution of the application.
- **Sink:** Places where an application can send information to other applications or to the internet, such as SMS sending or HTTP connection.

Information flow can be tracked in two ways [38]: implicit and explicit. On explicit the information is passed directly from one tainted variable to another. Whereas implicit

information flow, the value from a tainted variable affects the value of another variable indirectly. Figure 2-3 shows an example with implicit and explicit flows in a code snippet. Line 2 is an example of explicit flow because variable x receives the value from y directly while lines 4 and 6 are examples of implicit flow because the value that the variable send receives depends on the if statement.

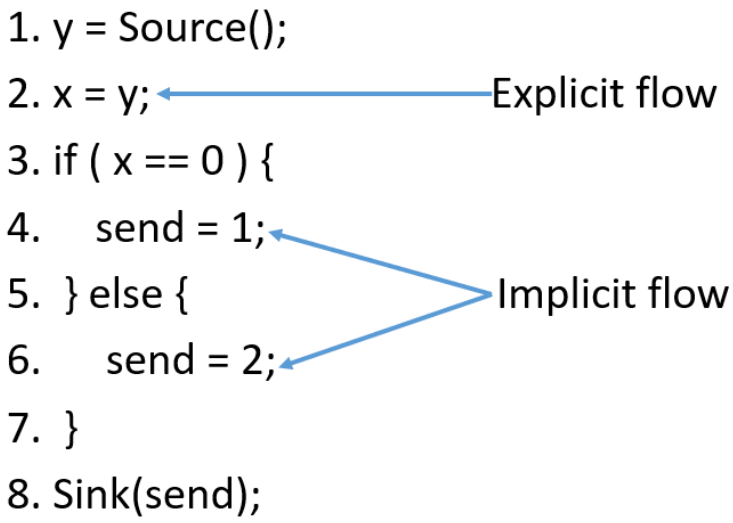

Figure 2-3: Explicit and Implicit Flow Example in a Code Snippet.

Taint Tracking can be static or dynamic. In static analysis the tracking works by simulating the execution of an application, usually creating a Control Flow Graph (CFG) that attempts to find possible paths from sources to sinks by simulating the execution flow of the application, it tracks at each instruction which taints are influencing the variables of the app. Figure 2-4 provides an example of a code snippet and a representation of this code as a graph.

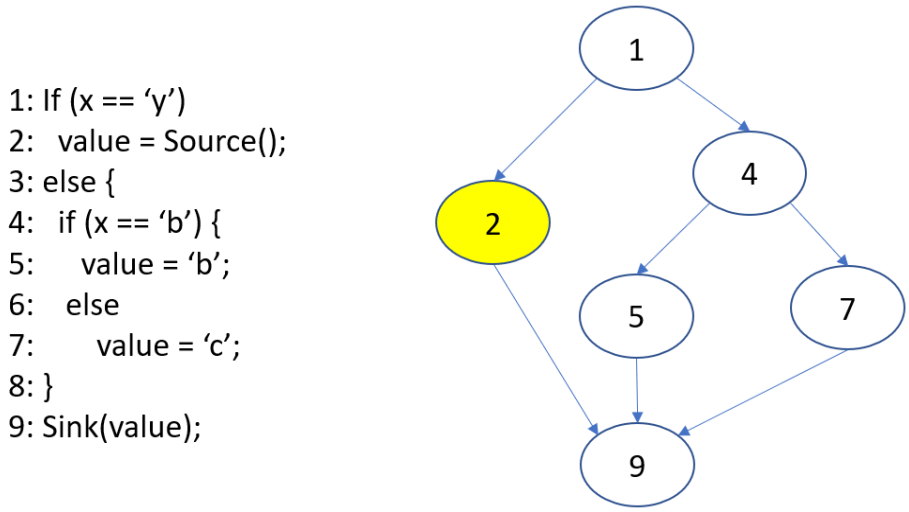

Figure 2-4: Static Analysis Example.

The goal of the graph is to cover all the possible paths that the code snippet can go through. For example, from line 4 the code can go to line 5 or line 7 and the graph is covering both situations and the variable value is going to be tainted only if the first if statement of the code is true. Among the Static Taint Tracking benefits are the capability of analyze all possible executions paths of the program, point the code where issues occur, detect problems in the early stage of development and works well with explicit flow [39]. In contrast, it is challenging to apply Static Taint Tracking for implicit flows because the values of the variables are affected indirectly. For example, Android applications can be built using reflection, i.e. it is possible to instantiate new objects, invoke methods and get/set field value at runtime. Besides that, Static Taint Tracking can have overtainting problems.

Dynamic Taint Tracking is the process of executing an information flow tracing while a program is being executed. This type of tracking learns the properties of the application by executing it in a controlled environment and examining the program state during and after execution. CuckooDroid [40] is an example of Dynamic Taint Tracking environments for Android apps. Figure 2-5 shows Cuckoo architecture.

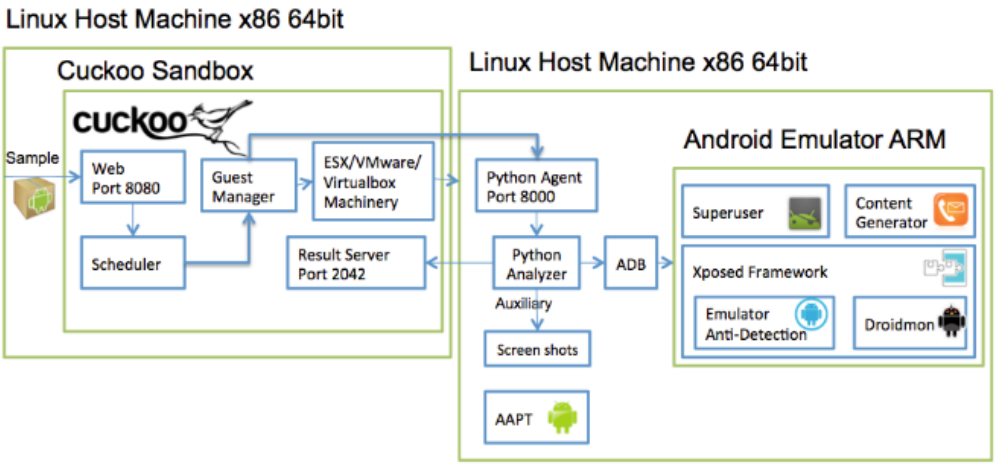

Figure 2-5: CuckooDroid Architecture.

Cuckoo is composed of a host machine and a couple of Virtual Machines (VMs), called guests, that execute the applications that are being analyzed. Each app analysis is launched in an isolated and new VM that contains a Linux virtual machine to run an Android emulator. Among Dynamic Taint Tracking benefits are the capability to analyze the state of the applications during runtime, be sure that a problem exists if it is found and also validate Static Taint Tracking results. However, Dynamic Taint Tracking is too expensive to run, it is input dependent and most of the time cannot explore all the paths that an app can execute. This thesis applies Static Analysis to choose the sensitive static API calls that could lead to a data leakage. These API calls are used in a Machine Learning model that classifies applications into malign or benign. Besides that, the sources and sinks collected are used to create reports with the possible behaviours inside the Logical Predicate Mapping (LPM), more information about the development in Chapter 4.

2.3. Machine Learning

Another computer science field used in this thesis is Machine Learning, that is defined as the science of programming computers so they can learn from data [42]. For instance, a Machine Learning program can learn how to flag malwares. To do that it needs a previous set, called the training set, with samples of malign and benign applications flagged by users to train the algorithm and create a model that will later be used for classification. Figure 2-6 shows the Machine Learning process that can be divided into five steps. Problem Definition is the first step, here some questions need to be answered such as: What business problem are we trying to solve? and How can this problem be modeled as a machine learning problem? if it is not possible to answer these question then it is not a Machine Learning problem. To model the Machine Learning problem, a Machine Learning system type needs to be chosen, these types can be classified in categories based on [42]:

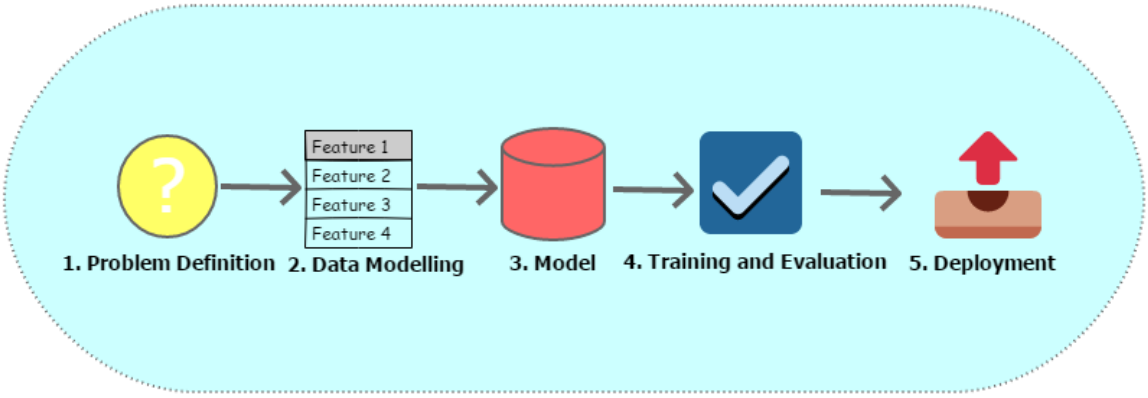

Figure 2-6: Machine Learning Process Steps.

1. Whether the model was trained with human supervision or not (supervised, unsupervised, semisupervised and Reinforcement types).
2. Whether the model can learn incrementally (online and batch types).
3. Whether the model detects patterns or works by comparing data points in the training data (model-based and instance-based types).

This thesis focuses on the first category and on supervised learning, that is when the training data added to the algorithm has the desired solutions, called labels, the algorithm tries to learn the patterns that can lead to the labels. Typical supervised learning tasks are classification and regression. Spam filters are good examples for classification tasks where the model is trained with as much as possible examples of emails (spam or not) and it must learn how to classify new samples.

On regression tasks the model tries to predict a numeric value. For instance, it could try to predict the value of a house given a set of features (size, location, and others). In this case the system should be trained with as much as possible examples of houses,

including their predictors and the labels. Examples of supervised learning algorithms are K-Nearest Neighbors (KNN), Linear Regression, Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVMs), and others. This thesis focuses on classification. The second step of the Machine Learning process from Figure 2-6 is Data Modelling, the data collected is important because the quantity and quality of the data is determinant to get a good Machine Learning model. If the data is insufficient, nonrepresentative, polluted, noisy or has irrelevant features the model may not have good results.

There is also a problem with overfitting and underfitting. Overfitting is when the model performs well during training but it does not generalize well and underfitting is the opposite and occurs when the model is too simple [42]. Therefore, the data really needs to be good enough to build the machine learning model.

The data format can be structured or unstructured and can contain static or streaming data types. Structured data format can be represented as columns and rows. For instance, patients records, movies database and so on while unstructured data can be images and audios. Static data is information that does not change such us customers purchase history while streaming data is information that is constantly being updated such as users' location. This thesis focuses on structured and static data. After the data is collected, the features need to be chosen and the dataset prepared to be used, most algorithms require a specific data format to be used. With the problem defined and the data prepared the next step is to choose the model and start the training, the best model choice depends on the problem and on the data collected.

Bad performance on training data means the model has not learned anything and it needs to be improved, changed or more data needs to be collected while poor performance on test data means the model did not generalize well and probably had an overfitting. Some metrics were created to evaluate classification problems, they are listed below:

1. False negatives (FN): Considering a malware detection machine learning model. False negatives correspond to the number of malicious applications that were incorrectly classified.
2. False positives (FP): This metric is the opposite of false negatives. For instance, a benign application that is classified as malware by the model.
3. True negative (TN): Number of benign apps that were successfully classified.
4. True positive (TP): Number of malware apps that were successfully classified.
5. Precision (P): This metric is the division between the number of true positives divided by the sum of true positives and false positives so it is related to the relevancy of the results, a model that has no FP has a precision of 1. Formally it is $P = \frac{TP}{TP+FP}$
6. Recall (R): This metric is the division between the number of true positives divided by the sum of true positives and false negatives so it is related to how many

applicable results are returned, a model that has no FP has a recall of 1. Formally it is $R = \frac{TP}{TP+FN}$

7. F1-Score (F1): This metric is the harmonic mean between Precision and Recall, the closer to 1 is better. Formally it is $F1 = 2 \times \frac{P \times R}{P+R}$
8. Receiver Operating Characteristic (ROC) curve: The ROC curve is a plot comparing TP and FP rates.
9. Mean Absolute Error (MAE): Average difference between the model results and the actual values.
10. Root Mean Square Error (RMSE): This metric is the value of the square root of the average of squared differences between the model results and the actual values.

The last step is the deployment, the act to upload the model inside an application to use. A malware detection machine learning model was trained and deployed to be used in this thesis, more information about the development is in Chapter 4.

2.4. Probabilistic Topic Models

Induced by the need of new techniques for efficient text information extraction a new area of research emerged in 2003, called probabilistic topic models [43] with the goal to discover hidden thematic structures in a large collections of documents and find out similar topics, how they change over time and how they are connected. Initially, these models were proposed to be applied to textual documents only, but since it had good results the applicability was also explored in other data types with discrete attributes, such as images, graphs and others.

The idea behind topic models is to discover latent patterns that are meaningful in the relationship between documents and terms (words) [43]. For instance, a model can be used to rank a set of terms or documents as important for a particular theme. Probabilistic models simplify the exploration of large number of data through topics discovery, topics are structures with semantic values that form sets of words that often occur together. When these sets are analyzed they give clues about a theme or subject that occurs in a subset of the documents. For instance, movie synopses can be organized and grouped up by applying topic modeling to discover theme-based subsets such as comedy, horror, adventure, and others. This thesis uses Topic Models to group up the applications with similar goals.

Some techniques have been proposed to evaluate probabilistic topic models. Most of them use human intervention such as topics interpretability that requests a person to verify the top-k probability words from the topics to check if they are meaningful and topic/words evaluation when a random word is added to a single topic and a person has to look at the topic and discover the world. Other techniques do not use human

intervention such as Perplexity and Topic Coherence. Perplexity measures the log-likelihood of a held-out test set, in other words it measures how well the model represents unseen data [44]. For a test set of M documents the Perplexity is obtained through the formula:

$$p(D_{test}) = \exp - \frac{\sum_{d=1}^{M} \log p(W_d)}{\sum_{d=1}^{M} N_d} \qquad \text{Equation 2-1}$$

where, W_d represents the worlds from each document d and N_d is the number of words in each document d.

However, Perplexity does not always correlate with semantically human interpretable topics and often it is totally not correlated. This limitation served as motivation for studies to model the human judgment, with that other techniques such as Topic Coherence emerged. Topic Coherence measures score a single topic by obtaining the degree of semantic similarity between the words with the highest score in the topic [45]. These measurements help distinguish between topics that are semantically interpretable and topics that are only artifacts for statistical inference.

Latent Dirichlet Allocation (LDA) is a popular probabilistic topic model technique and it is based on the generation of topics as Dirichlet distributions. In LDA a collection of composites is made up of parts where the composites are documents and the parts are words and/or phrases, through this an LDA model can classify unknown documents using a priori information.

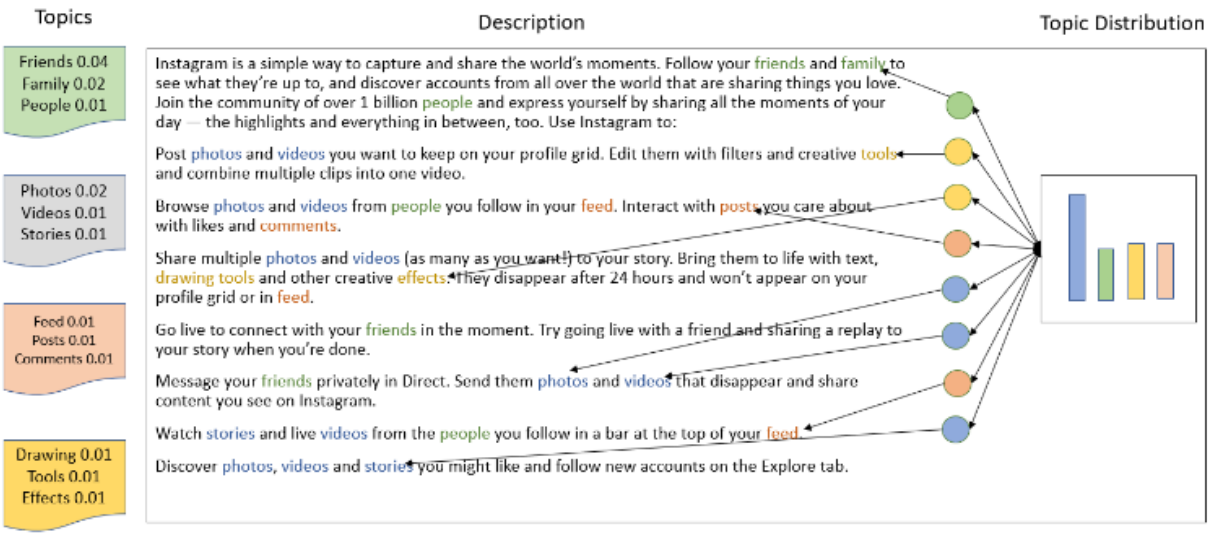

Figure 2-7: LDA Topic Distribution and Assignment in Instagram.

LDA model reflects the idea that documents are made up of various topics that are displayed in different proportions, each word in a document is obtained from one of the topics, where the chosen topic is selected from the per-document distribution over topics. This is the most distinctive characteristic of LDA, the documents share the same topics. However, each document exhibits them in different proportions. Figure 2-7 shows an

example with Instagram description, highlighting four topics, with blue being the one with the highest proportion. The generative process can be represented graphically through a Bayesian network as illustrated in Figure 2-8. On the network the vertices represent the variables while the edges represent the dependency relations. At the graphic model notation, a rectangle means that the variables are grouped up into a subgraph that repeats. The number of repetitions is represented at the bottom of each rectangle. Each element from the notation are described below:

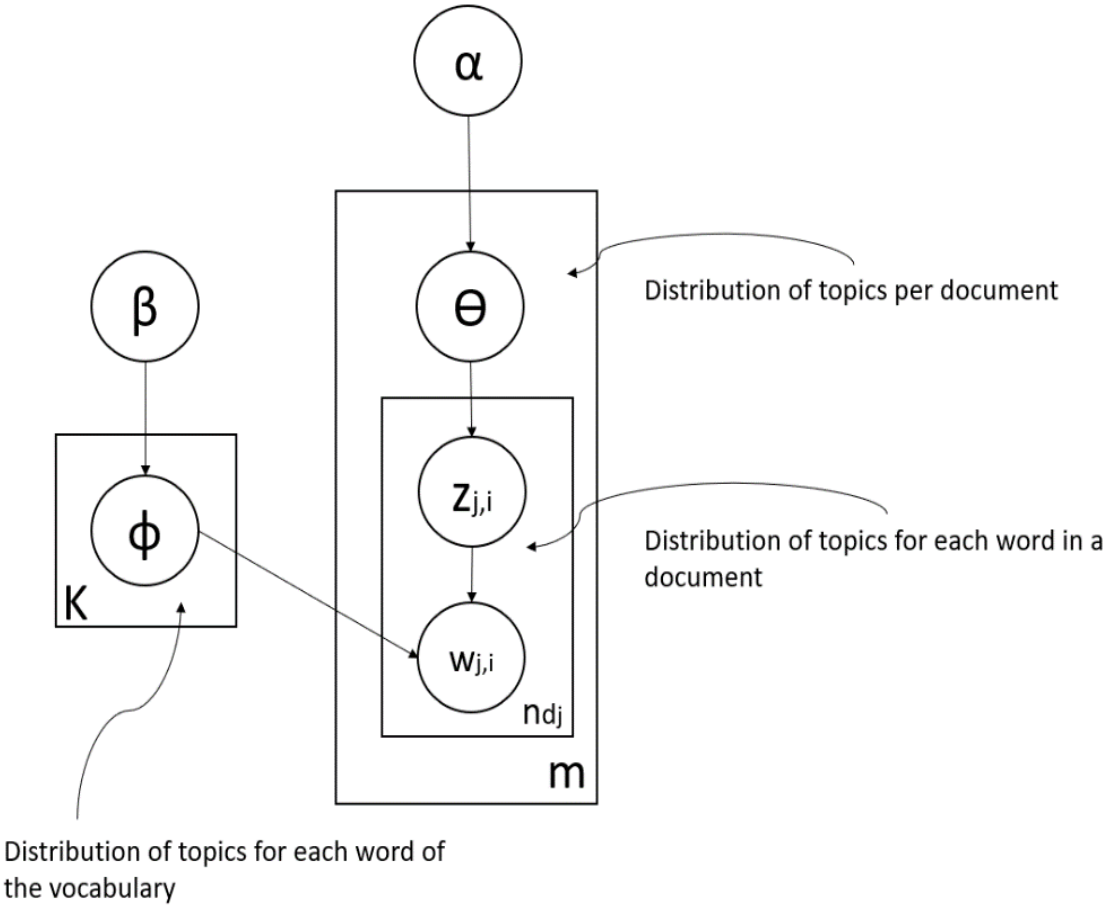

Figura 2-8: LDA Graphical Model.

- K – Number of topics.
- n – Number of words on the vocabulary.
- m – Number of documents.
- n_{dj} – Number of words in a document d_j, *where 1<=j<=m.*
- Θ – Topics distribution per document.
- ϕ – Topics distribution over all the words in the vocabulary.
- Θ_j – Vector with the topics proportion for a document d_j, where 1<=j<=m.
- ϕk – Vector with the words from the vocabulary proportion for a topic k, where 1<=k<=K.
- α – Priori from Dirichlet distribution, related to the document-term distribution.
- β – Priori from Dirichlet distribution, related to the topic-word distribution.

- w_i – Word from the vocabular where 1<=i<=n.
- $w_{j,i}$ – Word wi observed on document dj, where 1<=j<=m and 1<=i<=n.
- $z_{j,i}$ – Topics distribution associated to the word $w_{j,i}$ on document d_j, where 1<=j<=m and 1<=i<=n.

The model is organized hierarchically in 3 levels. First level represents the topics distribution in the set of documents. Second level represents the topic distribution for each document and last level makes possible the representation of a document as a mixture of topics. The hyperparameters α and β are used to calculate the topics and the words inside the topics. For instance, a higher value for α means that each document will have a higher mixture of topics and a lower value indicates a higher probability of documents with the mixture of few topics. In the same way, a higher value for β means that each topic will have a higher probability of containing a mixture of multiple words. While a lower value indicates that the topic will contain few words.

The variable ϕ_k is located at the second level for each topic k. Each vector ϕ_k forms a matrix ϕ of size n x K, where each line represents the words from the vocabulary and the columns represents the topics. The value from $\phi_{k,i}$ is the proportion of a topic k to a word w_i. The distribution of documents per topic is represented as a matrix Θ of size m x K, where the lines are documents and the columns are topics, one line from this matrix matches the topics proportion for a document dj of the list. The worlds level contains the variables $z_{j,i}$ and $w_{j,i}$. These variables are shown for each word wi in each document d_j. The variable $z_{j,i}$ is the assignment of a topic k (1<=k<=K) for a word w_i of a document d_j. In summary and following the description LDA generative process executes the steps described below:

1. Shows K multinomials $\phi_k \sim Dir(\phi_k;\beta)$, for each topic k.
2. Shows m multinomials $\Theta_j \sim Dir(\Theta_j;\alpha)$, for each document dj.
3. For each document dj:
   i. Assign a topic for each $z_{j,i}$ from the Θ_j Dirilecht distribution.
   ii. Shows a word w_i from the distribution $\phi_{zj,i}$.

Based on the generative process and looking at the relationship of dependence between the variables of the model, it is possible to describe the probability of all the latent variables of the model given the information a priori [46].Transcribing these probabilities the following joint distribution is created:

$$p(z,w,\emptyset,\theta|\alpha,\beta) = \prod_{k=1}^{K} p(\emptyset_k|\beta) \prod_{j=1}^{M} p(\theta_j|\alpha) \left(\prod_{i=1}^{V} p\left(z_{j,i}\middle|\overrightarrow{\theta_j}\right) p\left(w_{i,j}\middle|z_{i,j},\emptyset_{z_{j,i}}\right) \right). \quad \text{Equation 2-2}$$

Taking into account the observed and unobserved variables and their dependencies, the goal is to discover topic assignments for the documents, topics distributions per topics and topics by terms. So, the biggest computational problem for LDA is to infer

p(z,ϕ,Θ,|w,α,β), where w is all the words observed in the list of documents. From Bayes theorem, it is possible to formulate the probability of p(z,ϕ,Θ,|w,α,β) as the calculation of the a posteriori of LDA like it is shown below:

$$p(z, \emptyset, \theta|w, \alpha, \beta) = \frac{p(z, w, \emptyset, \theta|\alpha, \beta)}{p(w)}. \qquad \text{Equation 2-3}$$

The joint distribution (Equation 2-2) is the numerator and the denominator is the probability of the observed data. Therefore, the central computational problem can be solved by inferring the posteriori probability of the whole model, described in Equation 2-3. This thesis uses LDA to get the set of descriptions from the applications that belong to the same store category, discover their latent topics and find out the topics distributions that are more similar to the description of the application being evaluated in terms of functionality. Chapter 4 has more information about the development of the algorithm.

2.5. Final Considerations

This chapter presented the concepts necessary for the understanding of this thesis starting with the Android operational system, its architecture, application structure and security features describing how the technology works and how developers could apply best practices to avoid security and privacy issues such as data leakage and system exploitation. After, the techniques and algorithms used and integrated in the development of this thesis prototype were explained. Starting with Taint Analysis, a technique that is used to track information, explaining the types, how they work, their benefits as well as its limitations.

The second technique explained is Machine Learning, a field from computer science that is used to solve problems through knowledge acquired from data, explaining what it is, the steps of a Machine Learning process as well as some challenges that needs to be addressed to use Machine Learning. Finally, Probabilistic Topic Models, a technique that is used to discover hidden topics inside a set of documents, is explained with focus on Latent Dirichlet Allocation (LDA).

This thesis used LDA with the goal to receive a description from a target application. The topic distribution from the application description is obtained and later compared with the topic distribution from the set of benign applications descriptions from the same category as the target application to find out the ones that are most similar to the target app in terms of application functionality. Next Chapter presents some research about Android mobile application recommendation works that suggest applications and/or permissions configuration to users but also verifies some security or privacy aspect.

Chapter

3 Related Works

This chapter presents works related to recommendation systems usage inside the Android environment that also focus on security and privacy. To better organize this Chapter the works are divided into approaches that recommend permissions configuration and approaches that recommend Android applications installation.

3.1. Approaches that recommend permissions configuration

This category contains works that analyze some aspects of a group of applications to recommend which permission configurations are most suitable for the app being evaluated.

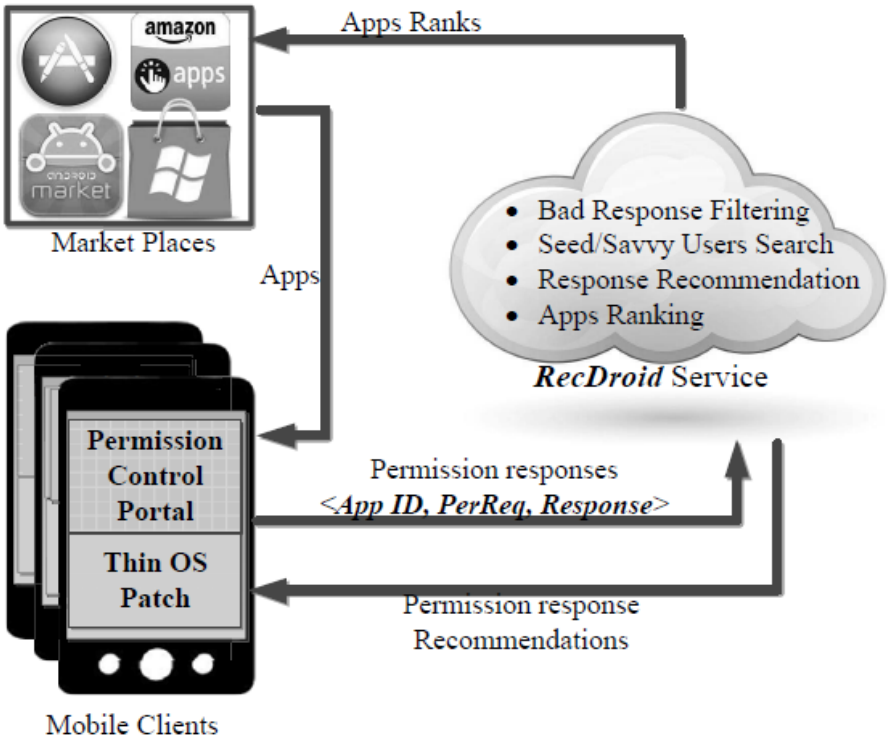

Figure 3-1: RecDroid Service Overview.

For instance, Rashidi et al. [47] created RecDroid with the goal to control Android applications permissions in real time through recommendations of permissions configuration from expert users (crowdsourcing) who use similar apps. With this approach, regular users will be able to make correct permission granting decisions based on the choices made from the expert users avoiding security issues. Figure 3-1 shows an overview of RecDroid which is composed of a patch at the Android operational system and a service to report permissions suggestions. RecDroid users can choose between two types of modes during the application installation: trusted mode and probation mode. In probation mode the application has to request permission to access the sensitive resources during runtime (e.g. Location, contacts). In trusted mode, all the requested permissions are granted whereas during probation mode RecDroid queries it services to compare the requested permission with a list of predefined dangerous perms to suggest if the permission should be allowed or denied.

RecDroid also recommends if an application should be accepted during installation based on its collected data. For instance, if an application was rejected a lot of times RecDroid recommends that this app should not be installed. However, the process of installing RecDroid inside a mobile device may be hard for regular users because the Android operational system needs to be modified. Also, since its recommendation system is based on user suggestions, some malicious users could give the system misleading answers to benefit a particular app.

Bao el at. [48] present APRec, a permissions configuration recommender based on the perception that the applications that execute the same Application Programming Interface (API) share similar features and also use similar permissions configuration. Figure 3-2 shows APRec framework.

APRec starts gathering applications APIs to create representative feature vectors. After, the distance between the vectors is calculated using a cosine similarity approach. Finally, a recommendation score for each permission using a collaborative filtering approach is calculated and the permissions with the highest scores are suggested to the users. APRec is entirely focused on API calls and there are certain features in Android that can be accessed through more than one resource. In addition, there are API calls that have been deprecated in newer versions of Android and have been replaced by new methods. For instance, the Android Lollipop BluetoothAdapter had some deprecated methods that were changed by BluetoothLeScanner, which may influence the results of the approach.

Rashidi et al. [49], same authors from RecDroid, proposed another framework called DroidNet, that also uses decisions made from expert users to provide recommendations on whether to reject or accept an application permission configuration. DroidNet runs the applications in probation mode to request recommendations on whether to allow or deny the requested permissions configuration based on decisions from expert users. The main difference from DroidNet and RecDroid is the addition of a rating algorithm to seek expert users based on transitional Bayesian inference. However, DroidNet shares the same

restrictions as RecDroid, since installing the approach inside the phone may be difficult for regular users as the Android operational system needs to be modified.

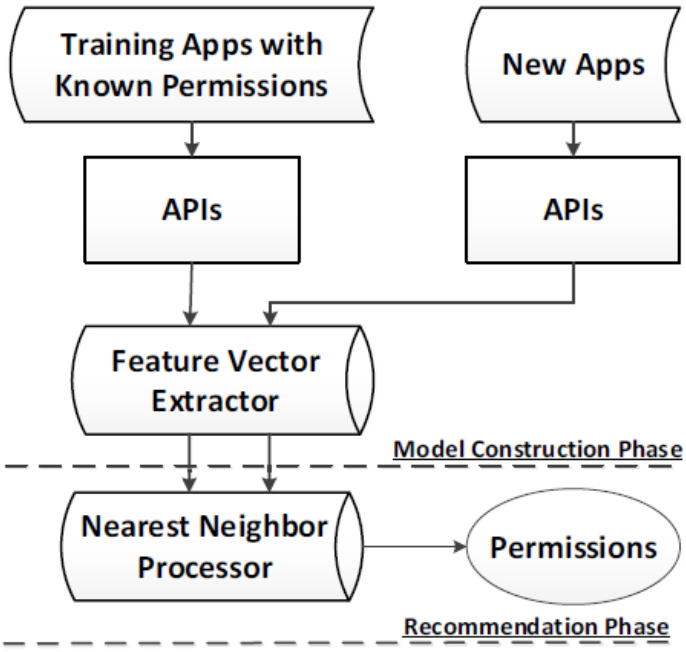

Figure 3-2: APRec Framework.

Latifa et al. [50] present PermisSecure, which works by executing static and dynamic analysis of an application before installation to detect applications that request dangerous or unnecessary permission configurations. The analysis is divided into two levels: First level checks the permissions and warns the user if they are dangerous. If necessary, it goes to the second level that checks the application code and shows a warning if there is anything suspicious. PermisSecure may present performance overhead mainly caused by the dynamic analysis. Moreover, Android operational system modification is needed.

Shukla et al. [51] present PermissionChecker, an approach that uses the categories from Google Play to inform when an application that belongs to a determined category should receive certain permission configuration similar to most applications in that particular category. PermissionChecker is divided into three steps: data collection, data analysis and recommender. The data collection uses the apps category from Google Play to divide the applications into clusters.

The data analysis step applies a recommendation technique that determines the permission configuration that should not be given to an application. Then, the recommender accesses the apps/permission matrix to show the users which permission configuration should or should not be granted. However, PermissionChecker can sometimes cause runtime errors if it revokes a permission configuration related to the app normal operation.

Liu et al. [52] present PriVs, a permission configuration recommender based on a crowdsourcing approach. PriVs works by collecting permission configuration from applications and learning similarities based on privacy preferences on the applications. In addition, PriVs gathers user opinions about the permission configuration when the information is required. These opinions can be: agree, reject or agree only this time. PriVs does this in an attempt to improve its model. PriVs is composed of two components: a mobile application installed on the user device that is responsible for scanning all the other applications installed and to provide an interface to configure permissions configuration and receive recommendations. Second, a recommendation server is used to process data collected from the users and generates the suggestions. One restriction of PriVs is the need to install the app with root access.

Liu et al. [21] present PriWe, a system that also uses crowdsourcing to suggest privacy permissions configuration based on users that previously used the apps on their mobile devices. PriWe has two components: An application located inside the mobile device that gathers privacy settings and a recommendation server that analyzes the crowdsourced data to generate suggestions. PriWe mobile device application has some functionalities such as scanning installed apps, browsing permissions configuration and applying permission suggestions from the recommendation server. Just like PriVs, PriWe requires that its mobile device application runs at the system level to gather all the privacy settings, for which it needs root privilege.

Do et al. [53] propose a work that studies social network applications and removes the permissions that may cause privacy issues. Their approach works decompiling the APK file to access the app source code to remove the permissions that may cause a privacy problem. Then, the app is recompiled to be used inside the mobile device. However, applications may not have their full functionalities and/or crash after the permission removal. Also, repackaging an app may be considered a malicious behavior by app stores.

Ali et al. [54] present Auto Android, a framework that enables users to customize permissions configuration at install time and during runtime. Auto Android has a machine learning layer that checks applications categories and permissions to suggest which permission should be configured to avoid data leakage. As in previous works, Auto Android requires Android operational system modifications, this is a hard task for regular users.

Oglaza et al. [55] propose Kapuer, a system that uses machine learning to understand user preferences and create rules that permit or deny a permission configuration request. Kapuer can also create high-level permission configuration rules such as “Applications from Social category can have local access to user data” which is a global rule for all the applications from the social category. A drawback is that Kapuer has to be installed on rooted mobile devices.

3.2. Approaches that recommend applications installation

This is the category most related to this thesis. It consists of works that analyze some aspects of a group of applications to recommend if the application being evaluated should or should not be installed on the mobile device.

Dang et al. [56] present TruBeRepec, a mobile recommendation system that generates application trust and reputation scores according to individual collected information. Figure 3-3 shows TruBeRepec architecture that corresponds to a mobile application and reputation service provider (RSP) running as a server. The information collected can be from three different categories related to: normal application usage (Using Behavior - UB) such as usage time and frequency; usage after application problems/errors occur (Reflection Behavior - RB) and Correlation Behavior (CB), which is information taken from similar apps.

The collected information is used to calculate an individual trust value between 0 and 1. After that, the reputation score value is obtained by aggregating each trust value. Finally, a list with installed applications and their scores can be seen in the mobile application. TruBeRepec runs continuously to monitor application use on the mobile device, increasing battery consumption. In addition, it tries to find similar applications only by comparing application permissions. This can raise false positives as two applications that are completely different may use the same permissions.

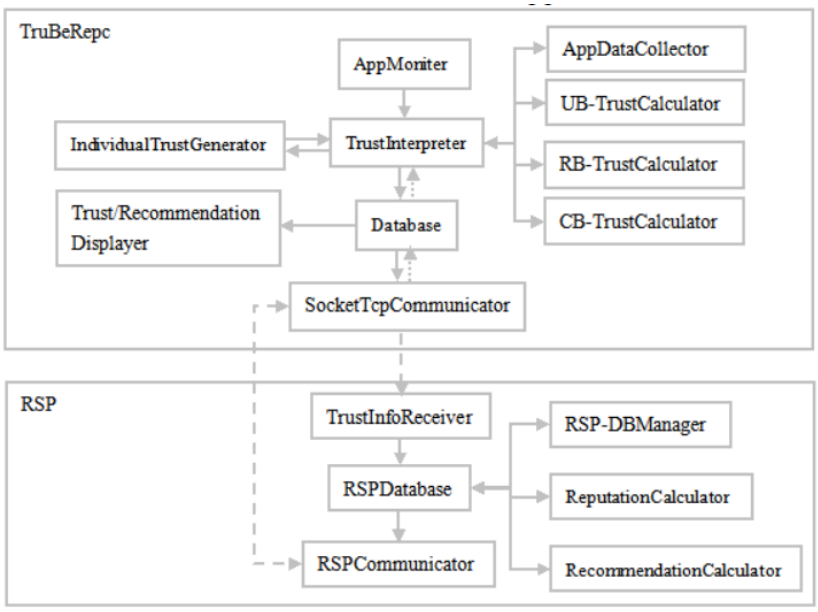

Figure 3-3: TruBeRepec Architecture.

Su et al. [57] propose a network traffic analyzer from Android applications that have similar functionalities to suggest Android apps installation. The top 100 Android applications from each of Google Play most popular categories, such as communication and entertainment were downloaded, executed and had their traffic cost extracted to create a dataset. From such information, a traffic-cost based recommendation algorithm

is used to suggest applications based on the traffic cost rate, percentage of required app traffic and some regular metrics that are also used in other works such as user ratings, number of downloads. However, one restriction for this approach is that to ensure that the network traffic generated for each single application is good enough, each app has to be individually installed and executed on the mobile device and, in reality, this scenario is infeasible and does not occur.

Rustgi et al. [58] present DroidVisor, a system that provides custom application recommendations to users based on security, similarity, popularity and ratings. To compute these scores, DroidVisor first selects apps that belong to the same category of the app being evaluated. After that, a keyword filtration approach is applied to select only the most similar applications. Finally, the final set of applications is used to calculate security, popularity and usability points (scores) based on the features extracted. The results are listed to the users so they can decide if the application should be installed or not. A restriction of DroidVisor is related to the security risk analysis of the apps that is based only on the permissions set.

Gomez et al. [59] propose a mobile application recommendation system for store administrators that suggests the removal of buggy (error-suspicious) applications through the investigation of the correlation between permissions configuration and errors reported from users in applications reviews made in the stores. Figure 3-4 depicts the system overview. The system begins extracting the Android application metadata and users reviews from Google Play Store. Then, a generative statistical model is used to group up only the applications that are error suspicious. Finally, error-suspicious applications and permissions are correlated to build a model that is used to suggest the apps that should be removed.

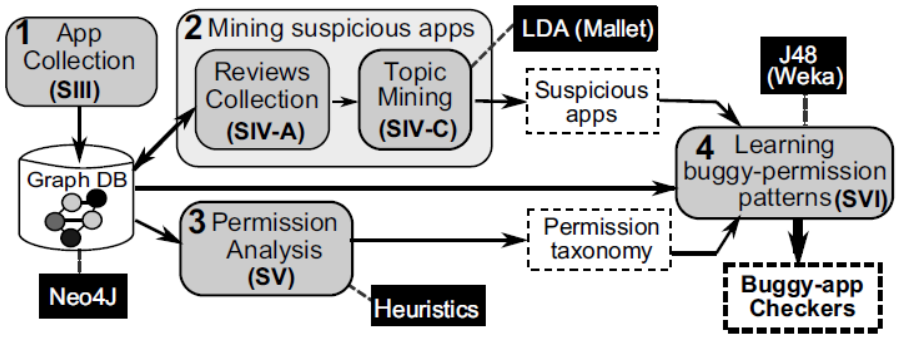

Figure 3-4: Buggy Apps Overview.

Since the system is based on user application reviews a malicious user may provide false information to harm the app popularity. In addition, applications can crash for other reasons that are unrelated to permissions configuration, such as an out of memory error.

Jisha et al. [60] propose a mobile recommendation system that evaluates popularity and security metrics based on user apps ratings (stars assigned to the applications) in the

Play Store and applications permissions configuration so that a user can choose which application to install on their mobile device. Figure 3-5 has the system architecture.

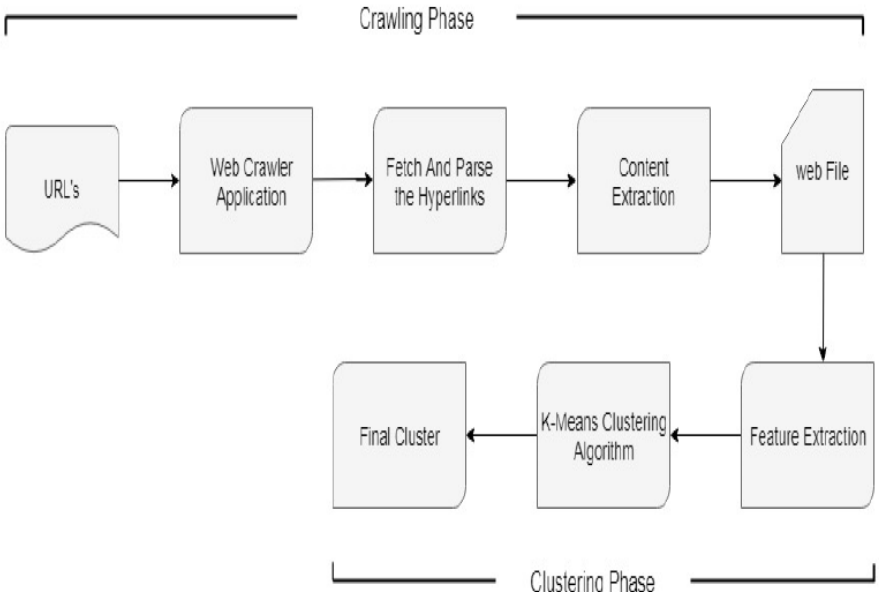

Figure 3-5: User Rating and Permissions Architecture.

The system works by crawling applications from the official store and extracting user ratings to create clusters based on rating values with high and low scores. Next, permissions are extracted, and a security risk is calculated. This data is integrated into a user interface that shows the clusters with the overall rating score of each application, thus users can choose which application they do or do not want to install on their mobile device. The authors claim that applications with the lowest permission configuration scores are safer. However, this is not true since certain applications need permission configuration to perform functions, such as location, Internet access, and others. Therefore, an app with more permissions will not necessarily be a less secure or malicious app.

Zhu et al. [61] present a mobile recommendation system called SPAR that considers the applications popularity and permissions to suggest a list of safe applications to install. SPAR recommendations are done in three types: Security, Popularity and Hybrid.

For the security type, applications are ranked based on their risk scores and then ranked by popularity. For popularity, the applications are ranked based on popularity scores and after by risk. The hybrid principle considers both risk and popularity scores to find a middle ground between popularity and safety. Figure 3-6 shows the system architecture.

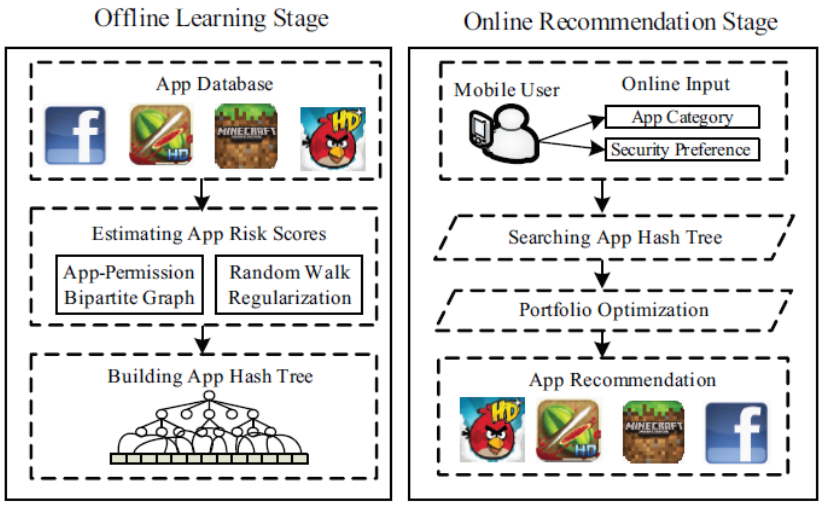

Figure 3-6: SPAR Framework.

SPAR is divided into two stages: The offline stage is used to calculate the apps risk scores and build a hash tree to perform faster searches, while the online stage is used to calculate popularity and applies a portfolio optimization technique to make recommendations.

Liu et al. [62] propose a work that recommends applications by calculating a trade-off between applications functionality and users privacy configuration. Their system starts modelling two vectors. One with user interest and another with application functionality, after it applies a function to calculate the functionality match score. After, other latent factor model is created to calculate the privacy respect using the users' privacy preferences and the apps private information (permission). Finally, the trade-off is obtained through a weighted sum between the match score and the respect score. Since their approach is based only on permissions configurations it has to assume that all the users completely understand the permissions system, which is untrue [21] so the privacy respect calculation may be difficult to get. Also, use the applications store ratings as a measure may also present problems since users may try to manipulate the rating system by using bots or by giving high ratings just to increase the application punctuation.

Zhu et al. [63] present a recommendation system that combines users preferences, privacy risks and apps functionality called AppURank. Their system starts calculating a Danger Coefficient (DC) based on privacy risk and application usage pattern. The first is obtained through applications permissions configuration and the second through application execution time. Larger DC indicates a bigger change for a privacy violation. After, user preference is obtained by representing each user's choices as a distribution of common preferences. This distribution suggests the preferences of users for each application category. After that, a topic model is used to create clusters between the apps and split them by functionality and usage context. Finally, AppURank is calculated by combining DC and users preferences. A possible drawback is that AppURank considers that the probability that an application has to access sensitive information is proportional to the time the application is used in foreground, ignoring one of the strongest features of

Android apps, which is the use of services in background for the execution of heavy tasks such as downloading a large amount of data. Also, AppURank privacy focus is fully based on permissions configurations.

3.3. Discussion

Table 3-1 summarizes the characteristics of each research. First column lists the approaches name. Second column lists from where the features were obtained that can be:

- **Collaborative filtering**: Category set when the approach uses features that are provided from other users. For instance, permissions configuration sugested from security experts.
- **Network traffic**: Category set if the approach uses the information from the network packages such as HTTP requests information.
- **Store features**: Category set if the work uses any feature extracted from the store such as application description, number of downloads, ratings, developer name, and more.
- **Permissions**: Category set when the related work uses the permissions configuration as features.
- **Device features:** Category set when the features are extracted from the device, such as application execution time.
- **Static features**: Category set when the source code from an Android application is analysed to extract features.
- **Dynamic features:** Category set when the approach uses a sandbox to execute the applications in a virtual environment to extract features.

Third column is related to the features that are used to focus any security aspect, these features can be:

- **User voting**: This category is set when the security evaluation is done through user voting. For instance, users can vote if an application that is being used is secure or not and when a new user tries to use the same application he will receive the votes from previous users and decide if he wants to run the app, creating a collaborative environment.
- **Network traffic**: This category is set when the features related to security are extracted from network traffic analysis. For instance, the traffic cost of the applications.
- **Permissions**: This category is set when the related work uses only the permissions to check if an application is safe or not. For instance, Android applications are divided into categories and most of the works use the permissions of the dangerous category to execute their evaluation.

Table 3-1: Related Works Overview.

Name	*Features*	*Security Feature*	*Recommendation*	*Focus*
TruBeRepec	Collaborative Filtering	User voting	Applications	Apps on the device
Network traffic	Network Traffic Store features	Network Traffic	Applications	Apps on the store
RecDroid	Collaborative Filtering	Permissions	Permissions	Apps on the device
DroidVisor	Store features Permissions	Permissions	Applications	Apps on the store
APRec	Permissions	Permissions	Permissions	Apps on the device
DroidNet	Collaborative Filtering	Permissions	Permissions	Apps on the device
Buggy apps checker	Store features Permissions	Permissions	Applications	Apps on the store
PermisSecure	Static features Dynamic features	Static features	Permissions	Apps on the device
PermissionChecker	Permissions	Permissions	Permissions	Apps on the store
PriVs	Collaborative Filtering	Permissions	Permissions	Apps on the device
User Rating and Permissions	Permissions Store features	Permissions	Applications	Apps on the store
SPAR	Permissions Store features	Permissions	Applications	Apps on the store
PriWe	Collaborative Filtering	Permissions	Permissions	Apps on the device
Permission Removal	Permissions	Permissions	Permissions	Apps on the device
Auto Android	Permissions	Permissions	Permissions	Apps on the device
Kapuer	Permissions	Permissions	Permissions	Apps on the device
Personalized Mobile App Recommendation	Store features	Permissions	Applications	Apps on the store
AppURank	Permissions Device features	Permissions	Applications	Apps on the device
Proposed System	**Static features** **Store features**	**Static features** **Store features**	**Applications**	**Both**

- **Static features**: This category is set when any feature extracted from the applications through static analysis is used to apply any evaluation about security. For instance, API method calls are extracted from the source code through static analysis.

- **Store features:** This category is set when any feature extracted from an application store is used to apply any evaluation about security. For instance, developer name, number of downloads and android version are good features to evaluate security.

The fourth column shows if the works recommends mobile applications or permissions configuration. To finish, fifth and last column depicts the approach focus that can be applications that are on the mobile device, applications on the store or both.

In general, the only aspect verified to check security or privacy in recommendation works that suggest Android applications is the permission configuration analysis. Most of the approaches do that because permissions can easily be retrieved from the Play store. Permissions are an important metric to validate the security of Android applications, however they are not enough, as shown in Section 1. Other features, such as static API calls and store features should be used to improve the result of the security evaluation inside the recommendation approaches and provide more reliable results to users. Moreover, most of the works listed do not seem to be concerned about the functionality of apps and produce recommendations considering different aspects such as Play Store category, popularity, usability and developers name. However, users need suggestions of apps with similar functionalities to perform their daily tasks.

Finally, privacy risk should also be calculated inside a functionality context as certain permission and/or API call may be considered dangerous in one context but benign in others. The bibliographic study carried out in this thesis demonstrates that although there are some methodologies proposed to evaluate security and privacy aspects in Android recommendation systems, much work is still needed to address all the problems that were found. Next Chapter presents a functionality-based recommendation system that aims to mitigate these issues.

Chapter 4

Mobile Application Recommendation System Overview

As previously mentioned, the increased use of mobile devices made users change the way they usually execute their daily tasks. Previously, when performing tasks such as a bank payment or an email sending users preferred using personal computers (PC), nowadays these tasks can be done with mobile devices. However, the success of mobile devices usage has led malicious software developers that used to create programs for personal computers to switch to creating applications with malicious intentions for mobile devices.

Due to the damage caused by malicious mobile applications, tools have been created to evaluate the applications and determine if they are malicious or not in an attempt to protect users and apps store. However, most of these tools only warn the users if the application being evaluated is malicious. Simply detecting that an application is malicious is not enough, more information about the threat, the application, and the mobile device state is needed so users can prioritize mitigation actions [64]. Furthermore, if an application is malicious users would need suggestions of substitute applications with similar functionalities as the evaluated app to execute their daily tasks. This objective is achieved through recommendation systems. However, as highlighted in Chapter 3, current recommendation systems have some limitations.

The rest of this Chapter is organized as follows. Section 4.1 has an overview of the proposed system. Section 4.2 explains the features extraction process, elucidating how the features were chosen, from where and how they are extracted and how they are represented. Section 4.3 explains how the classification is made through the training and creation of models through machine learning algorithms execution. Section 4.4 depicts how to recommend Android applications that are classified as secure and have similar functionalities as the app that is being evaluated. Finally, Section 4.5 describes how LPM is created.

4.1. Mobile Application Recommendation System

This section describes the stages of the functionality-based system architecture proposed to evaluate and recommend mobile applications in Android environments considering security, privacy, functionality, popularity, and usability metrics obtained through information extracted from the applications. The system can be formally defined as:

Definition 1 (System Statement). *Given an application app from category c, and a set of apps S = {s}, which contains a set of store features {f_i}, app permissions {p_1}, static API calls {m_i} and descriptions {d_i} the goal of the system is to evaluate app based on security, privacy, popularity and usability metrics to build a list of recommendations with applications classified as benign from S that are from the same category c and have similar functionalities based on their descriptions.*

To reach this definition some issues should be considered as:

1. How to extract features from the store?
2. How to extract permissions and static API calls from the application files?
3. How to properly evaluate security to make recommendations?
4. How to build the machine learning model?
5. How to recommend applications based on their description?
6. How to evaluate privacy?
7. How to obtain popularity measures?
8. How to obtain usability measures?

Figure 4-1 provides an overview of the proposed system, which consists of three stages (Feature Extraction, Classification and Recommendation). The system data entry can be done in two ways: Uploading the application itself (.apk file) or by providing the URL of the official Google Play store. Evaluating an .apk file is necessary because not all apks are directly installed from the official store, there are several Android application stores such as [65], [66], [67]. In addition, there are manufacturers and companies that directly send their application apks to costumers to be installed directly without being uploaded in any store.

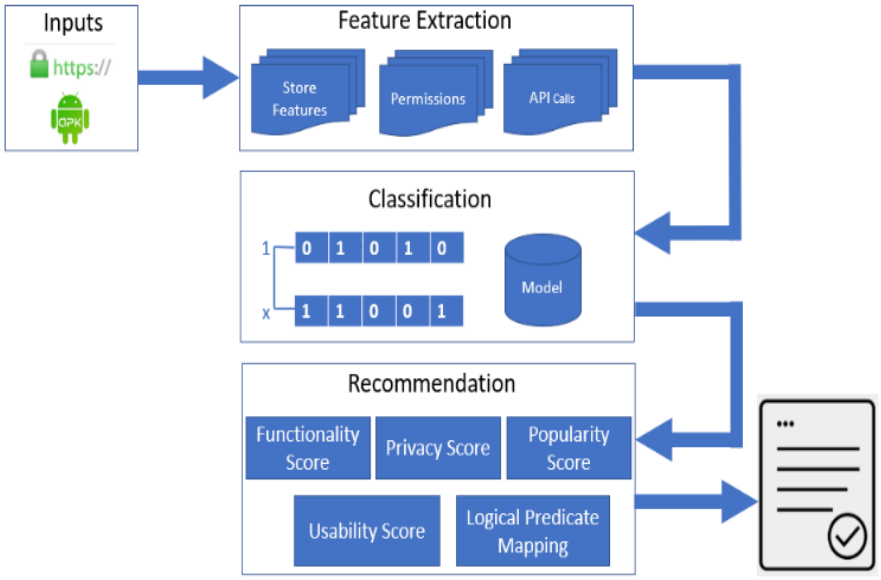

Figure 4-1: Proposed System Overview.

For the feature extraction stage, an APK is unzipped to get its main files (Android Manifest and .dex files). These files are used to extract the permissions and static Application Programming Interface (API) calls. All this data is grouped to create a relevant feature vector that represents the app. The feature vectors are then used to create a classification model (classification stage) that can predict if an app is either malicious or benign.

After classification, the recommendation process starts with only applications classified as benign and proceeds as follows: If the app being evaluated exists in the official Play Store, a list of suggested apps is obtained along with a features set to calculate privacy, usability and popularity scores. Otherwise, the user provides the category and description from the app being evaluated and an algorithm is used to retrieve the descriptions from each application that belong to the category of the app being evaluated. These descriptions go through a topic analysis process that determines which description is most similar to the description of the application being evaluated, to calculate a functionality score and decide which application is going to be suggested. Such score is later listed among the usability, popularity and functionality-based privacy scores and ranked through weights that are provided from users inside the system interface.

In addition, the proposed system employs an analysis summarization called Logical Predicate Mapping (LPM), which is the process of distilling the most important information from the descriptions, permissions configuration and API calls analysis to produce an understandable and abridged version for a particular user.

4.2. Features Extraction

Features extraction is an essential task in the proposed recommendation system since the features are the main factor ensuring the proper operation of a machine learning model. By studying the Android operational system security model to understand the permissions system, reading several works related to recommendation and malware classification such as [17], [58], [68] and studying the main methods (API calls) used to access user sensitive information, three types of features were chosen, as shown in Figure 4-2.

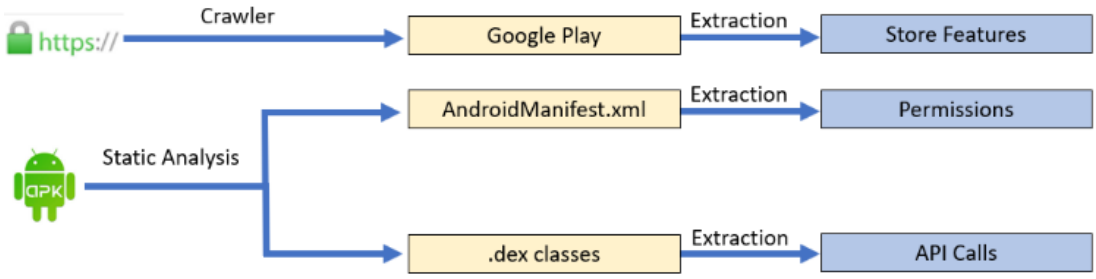

Figure 4-2: Features Extracted from an App.

The store features are obtained through a crawler that visits a store, i.e., Google Play. Table 4-1 lists the store features with a description of the feature.

Table 4-1: Examples of Features Extracted from App Store.

Feature	Purpose	Description
Category	Similarity	Category of the app such as education, finance, events and others.
Android version	Security	Version of the Android operational system.
Developer name	Security	Name of the company or person that created the app.
Number of downloads	Popularity	Number of times that the app was downloaded.
Rating	Usability	Score assigned by users to the app, can be from 1 to 5 stars in case of the Play Store.
Size	Security	Size of the app inside the mobile device.
Description	Similarity	Description of the app functionalities.

Static API calls were chosen based on a set of operations executed in Android applications such as sending and receiving files, database operations, advertisement libraries, command execution, access to private information and others. In addition, to enrich the choices, an algorithm was developed to group up the most used API calls, as shown in Algorithm 4-1.

Algorithm 4.1: APICallsSelection

```
Input: listAPK;
Output: listChosen
1: contIMEI=0, contSMS=0;
2: for j -> 1 to listAPK do
3:     listAPI = extractCalls(listAPK[j]);
4:     for I -> 1 to listAPI do
5:         if (listAPI[i] == 'getImei') then
6:             contIMEI++;
7:         end if
8:         if (listAPI[i] == 'sendSMS') then
9:             contSMS++;
10:        end if
11:    end for
12: end for
13: listChosen = sortConts(contIMEI,...,contSMS);
14: return listChosen;
```

The input for Algorithm 1 is the list of applications listAPK = {app_1,…,app_x} that are stored in the database. For each application listAPK[j] (line 2), the algorithm extracts all the Static API calls and adds them to a list called listAPI (line 3). Next, each API call is compared with sensitive API calls and a counter is incremented if positive. For simplicity, only two cases are shown in the algorithm (lines 5-10). Finally, a list with the most used sensitive API calls is filled (line 13) and returned (line 14). Table 4-2 lists 35 API calls that were chosen based on the criteria explained along with a description of their purpose.

Other static features extracted from the system are the permissions. Permissions in Android applications have the goal of protecting the privacy of the user [32]. For instance, to access sensitive data (i.e., contacts or SMS) Android apps must request permission, as well as some system features like Internet. Depending on the type of feature, the system might automatically grant permission or ask the user to allow the request. Two primary protection levels defined by the Android operational system [32] are considered: Normal and Dangerous permissions. Normal permissions are used when an application needs to access data outside its sandbox, but the risk to the user privacy is considered very low [69]. For instance, if an application requires information from Bluetooth, Wi-Fi, Internet, wallpapers, and others. Table 4-3 presents a list of the normal permissions.

Table 4-2: Examples of Sensitive Method Calls.

#	Method	Description
1	*WriteObject*	Used to write an object to a stream.
2	*getLine1Number*	Returns the phone number.
3	*getContentResolver*	May be used to retrieve a list of contacts.
4	*loadUrl*	Used to load a webpage.
5	*Connect*	Gets an HTTP connection and opens a communication link.
6	*getDeviceId*	Returns the device IMEI.
7	*getSharedPreferences*	Returns and object that represents a preferences file.
8	*getExternalStorageDirectory*	Return a shared/external storage directory.
9	*openConnection*	Return an HttpURLConnection.
10	*getResourceAsStream*	Returns an input stream.
11	*sendTextMessage*	Sends an SMS.
12	*sendMultipartTextMessage*	Send a multi-part text SMS.
13	*getSignature*	Gets the signature value from a certificate.
14	*Decode*	Decodes a string.
15	*openFileOutput*	Opens a file for writing.
16	*getUrl*	Returns an URL value.
17	*sendSms*	Sends a SMS.
18	*getNetworkOperator*	Returns the registered operator.
19	*checkBluetooth*	Verifies if bluetooth is enabled.
20	*getLongitude*	Returns the longitude.
21	*getLatitude*	Returns the latitude.
21	*getImei*	Returns the IMEI.
22	*getVersion*	Returns Android version.
23	*getAccounts*	Returns all accounts.
24	*getLastKnownLocation*	Returns a location.
25	*getSystemId*	Returns an id.
26	*getAltitude*	Returns the altitude.
27	*getConnectionManager*	Returns a connection manager.
28	*getUserInfo*	Retrieves user info.
29	*getLocation*	Returns location.
30	*getEmailAddress*	Returns the email.
31	*getLocalIPAddress*	Returns an IP address.
32	*getApplicationInfo*	Returns info about a package.
33	*getSensorList*	Retrieves a list of available sensors of a type.
34	*Inmob*	Advertisement library.
35	*Admob*	Advertisement library.

On the other hand, dangerous permissions are used when an application needs to access sensitive data or resources that pose risks to private user information [69]. For instance, if an application requires accessing information from the contacts. In order to

add access to a dangerous permission, a user has to explicitly approve the request to the application.

Table 4-3: Normal Permissions Names.

Normal Permissions	
ACCESS_LOCATION_EXTRA_COMMANDS	ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE	BLUETOOTH
BROADCAST_STICKY	CHANGE_NETWORK_STATE
CHANGE_WIFI_STATE	DISABLE_KEYGUARD
GET_PACKAGE_SIZE	INSTALL_SHORTCUT
KILL_BACKGROUND_PROCESSES	MODIFY_AUDIO_SETTINGS
READ_SYNC_SETTINGS	READ_SYNC_STATS
REORDER_TASKS	REQUEST_INSTALL_PACKAGES
SET_ALARM	SET_TIME_ZONE
SET_WALLPAPER_HINTS	TRANSMIT_IR
USE_FINGERPRINT	VIBRATE
WRITE_SYNC_SETTINGS	SET_WALLPAPER
ACCESS_NOTIFICATION_POLICY	UNINSTALL_SHORTCUT
BLUETOOTH_ADMIN	WAKE_LOCK
CHANGE_WIFI_MULTICAST_STATE	RECEIVE_BOOT_COMPLETED
EXPAND_STATUS_BAR	REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
INTERNET	

Table 4-4: Dangerous Permissions Names and its Groups.

Groups	Permissions
CALENDAR	• READ_CALENDAR • WRITE_CALENDAR
CAMERA	• CAMERA
CONTACTS	• READ_CONTACTS • WRITE_CONTACTS • GET_ACCOUNTS
LOCATION	• ACCESS_FINE_LOCATION • ACCESS_COARSE_LOCATION
MICROPHONE	• RECORD_AUDIO
PHONE	• READ_PHONE_STATE • CALL_PHONE • READ_CALL_LOG • WRITE_CALL_LOG • ADD_VOICEMAIL • USE_SIP • PROCESS_OUTGOING_CALLS
SENSORS	• BODY_SENSORS
SMS	• SEND_SMS • RECEIVE_SMS • READ_SMS • RECEIVE_WAP_PUSH • RECEIVE_MMS
STORAGE	• READ_EXTERNAL_STORAGE • WRITE_EXTERNAL_STORAGE

In addition, dangerous permissions are organized into groups related to the device capability or features. For instance, if an application requests access to the group permission called CALENDAR, the application will be able to read and write information on the phone calendar. Table 4-4 lists the dangerous permission groups with each of their permissions.

4.3. Classification Model

The classification model is responsible for evaluating security in applications. Unlike previous recommendation works that do not use machine learning and only evaluate permission configurations, the proposed system adds a security layer before recommendations are made so only applications classified as benign can be suggested to users.

The features obtained in the previous phase can be used to train and create the machine learning classification model. Each application is represented as a feature vector where 1 means that the feature occurred and 0 means that it did not occur, except for the features obtained from the store since they can have multi-valued information. For instance, multiple values are needed to represent Android operational system versions. The model is created through a training set with labeled data $\{F_j, v_j\}_{j=1}^{N_{tr}}$, where $v_j \in \{0,1\}$ and F_j is a vector containing the features values. Therefore, the label v shows if the appplications are malicious or benign and the vectors F_j represents the applications features.

Table 4-5: Applications Features Vector Representation.

Application	Features	Feature Vector	Class
A_1	F_1,F_3,F_5	<1,0,1,0,1>	L_m
A_2	F_3,F_4	<0,0,1,1,0>	L_b

To clarify the representation, a single application A_i is formally mounted in the form (Fa_i, La_i) where Fa_i is the feature vector of an application A_i and La_i is the class label that can be (L_M, L_B). Table 4-5 shows examples where two applications (A_1 and A_2) are listed with their feature vectors composed of five features. Application A_1 is a malware (Lm class) that has three features (F_1,F_3,F_5) so its feature vector is <1,0,1,0,1>, while application A_2 is a benign application (L_b class) that has only two (F_3,F_4) features and its feature vector is <0,0,1,1,0>. With the feature vectors, any machine learning algorithm can be used to train and generate the classification model. State-of-the-art works that focus on malicious applications classification use from traditional algorithms such as Naive Bayes [70] and SVM [71] to deep learning algorithms like Deep Belief Networks [68] and Autoencoders [72].

4.4. Recommendation

Although it is interesting to provide secure application recommendations, it is important to establish a ranking for applications that consider user expectations related to functionality, privacy, usability, and popularity. To deal with this problem, this work defines scores to generate the recommendation ranking. Specifically, we define numerical measurements (scores) for functionality, privacy, usability and popularity. These scores can be used to provide an ordered classification (ranking) according to a given criteria. For instance, to recommend applications that only perform similar tasks, a higher priority is given to functionality score herein, the other scores (privacy, usability and popularity) are ranked based on weights provided by users at the system interface through a seek bar, the values can go from 0 to 1. However, when the evaluated app does not exist or is not found in the official store, only its functionality and privacy scores are calculated.

4.4.1. App Scoring System

Popularity score is calculated by retrieving the highest number of downloads from the list of most similar applications to that being evaluated, with the application that has the highest number of downloads being considered the most popular. Therefore, popularity score can be obtained from the following equation:

$$p = \frac{Nea*100}{Nmp}. \qquad \text{Equation 4-1}$$

where, p is the popularity of the application being evaluated. N_{ea} is the number of downloads of the application being evaluated and N_{mp} is the number of downloads of the most popular application.

In the official Play Store, users evaluate applications with stars, ranging from 1 to 5, with 5 being the highest. Thus, to help achieve a usability score, it is necessary to calculate the average of the revisions made by the users for each app inside the official store. The highest rated app is considered the easiest to use because the reviews are usually based on the graphical interface of the app. The average revisions of an app are given by:

$$ar = \sum_{i=i}^{5} \frac{i*toti}{total}. \qquad \text{Equation 4-2}$$

where, ar is the average rating ranging from 1 to 5, tot_i is the total number of star rating votes with the index i and $total$ is the total number of users that rated the app. After this calculation, usability score can be obtained from the following equation:

$$u = \frac{Aea*100}{Ahr}. \qquad \text{Equation 4-3}$$

where, u is the usability of the application being evaluated. A_{ea} is the ratings average of the application being evaluated and A_{hr} is the ratings average of the highest rated similar application.

Moreover, the functionality score is obtained through a topic extraction technique. Like explained before, a topic is characterized by a collection of words belonging to a fixed vocabulary [43]. For instance, a soccer scores application would have a topic consisting of words related to matches such as {scorecard, table, teams, league, goals, goalkeeper, lineup}, while a stock exchange application would have a topic consisting of words related to stock market such as {shares, market, brokers, bid, trading, arbitrage, dividend}.

To execute the Topic Extraction, the descriptions of applications that belong to the same category of the application being evaluated are acquired and put through a data preparation step to remove stopwords, punctuation, words with small size, lemmatization and other strategies. Afterwards, the topics are extracted from the applications descriptions and a probability distribution over the topics is estimated. For instance, an app description can have a probability of 0.9 to be related to a specific topic and 0.1 to be related to another topic. The sum of the probabilities will always be 1.0. Basically, this is a clustering approach that groups the applications with a high probability of belonging to a certain topic.

The algorithm chosen to execute this task is the Latent Dirichlet Allocation (LDA) that is a generative probabilistic topic model of a collection of composites made up of parts where the composites are documents and the parts are words and/or phrases [46]. LDA was chosen because it does not need any information from users to suggest applications, it uses only the applications descriptions (treated as documents), preserving user's personal data. To determine the most similar apps related to the app being evaluated, the topic distribution from the evaluated app description is compared to the topic distribution of the other app descriptions that belong to the same category using the Jensen-Shannon distance metric, which is the square root of the Jensen-Shannon divergence (JSD). Equation 4-4 describes the Jensen-Shannon divergence.

$$JSD\ (P||Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D\ (Q||M) \qquad \text{Equation 4-4}$$

where, P and Q are discrete distributions (topics distribution in this case), $M = \frac{1}{2}(P + Q)$ and D is the Kullback-Leibler divergence described in equation 4-5.

$$D\ (P||Q) = \sum_i^X P(i) log\ (\frac{P(i)}{Q(i)}) \qquad \text{Equation 4-5}$$

where, X is the total number of distributions and i is the index of an event. The square root of the divergence is the Jensen-Shannon Distance, the smaller the distance is the more similar the two distributions are. Besides that, a functionality-based privacy score is calculated only considering the applications within the same category that have similar functionalities. In this work, an application is considered similar if it presents a

functionality score higher than 70%, this percentage was chosen through preliminary tests performed in the proposed algorithm.

Applications that execute the same functionalities tend to require the same permissions and access the same API calls because they execute the same task. For instance, consider applications that requires GPS permissions and belong to the set off apps related to location functionalities. In this case, GPS permission settings are not included in the privacy score calculation. Thus, for the calculation of the privacy score, only the unusual normal and dangerous features of the applications are used. Equation 4-6 describes how the privacy score is obtained:

$$pr = \sum_{i=1}^{N}[(d_i * wd_i) + (n_i * wn_i)] + \sum_{j=1}^{M}[(cn_j * wcn_j) + (cd_J * wcd_j)] \quad \text{Equation 4-6}$$

where, pr is the privacy score of an app, N is the total number of permissions, di represents a dangerous permission with index i, wdi is the dangerous permission weight, ni represents a normal permission with index i, wni is the normal permission weight, M is the total number of APi Calls, cnj represents an API call related to a normal permission with index j, wcnj is the API call weight, cdj represents an API call related to a dangerous permission with index j and wcdj is the API call weight. In our approach, unlike other solutions, weights are not calculated considering the categorization made by Google (Normal and Dangerous), this information is used only for feature selection. Instead, weights are calculated based on the probability of the feature being requested on the apps that belong to the same functionality set. For instance, if a set with thirty apps has ten applications that use the GPS permission, the weight will be 0.33. However, if only 5 apps have the permission the weight will be 0.16. Weights are calculated this way because the more apps that use an unusual permission the greater the likelihood that the app can be installed by users and cause possible privacy issues.

After the privacy score from all the applications inside the list of similar applications is calculated, the app with the highest score is considered the most dangerous app and used in Equation 4-7 to normalize the results and calculate the score of the app being evaluated.

$$pr = \frac{Pea*100}{Pmd} \quad \text{Equation 4-7}$$

where, pr is the privacy score, p_{ea} represents the privacy score of the application being evaluated, while p_{md} is the privacy score of the most dangerous similar app.

Based on the scores, a rank is made to show users which applications are recommended for installation and use. As mentioned before, functionality score has the priority at the calculation to show only apps that have the same functionality and the other scores (privacy, usability and popularity) are ordered by weights based on values chosen by users, such weights have values between 0 and 1 that are assigned in the framework interface.

4.5. Logical Predicate Mapping (LPM)

LPM is created using the permissions and API calls that are correlated, this is done by mapping the words to create a sentence or clause containing verbs and statements about the permissions and API calls (defined herein as predicate).

The idea behind the LPM is that the user can better understand what the applications can do with their mobile device by requesting certain permission or accessing certain sensitive information and take mitigation actions. Figure 4-3 has an example with location data that may be leaked through SMS sending. The predicates are created through a pre-defined dictionary with sources and sinks following the pattern "App {A} accesses sensitive information {SO} and may be leaking data through {SI}", where A is the name of the app, SO is the list of sensitive sources that the app accesses and SI is the list of accessed sinks that could lead to a data leakage. As stated in Chapter 2, sources are places where an application can get the sensitive information and sinks are places where an application can send information to other applications or to the internet.

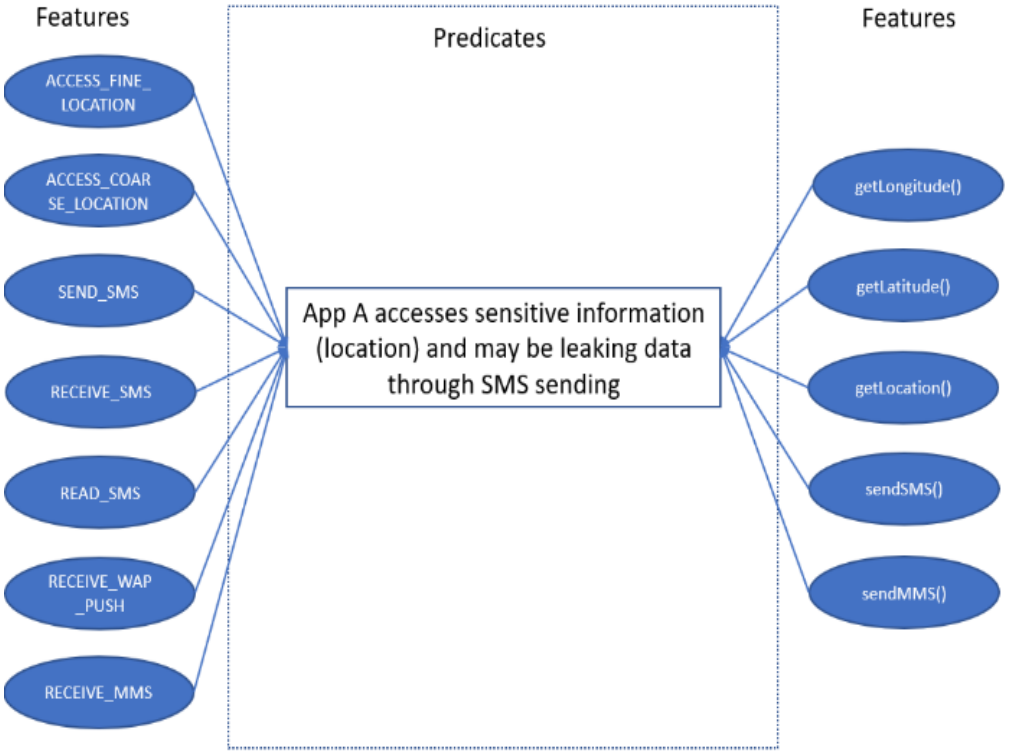

Figure 4-3: LPM Predicate Example About a Possible Data Leakage with Location and SMS.

The sources and sinks were chosen based on an analysis of the apps by [73] and also by reading works related to data leakage [74], [75], [76]. Table 4-6 shows examples of sources, while Table 4-7 has examples of sinks.

Table 4-6: List with Examples of Sensitive Information Sources That Are Used in LPM.

Sources	Permissions	API Calls
Location	ACCESS_FINE_LOCATION	getLongitude
	ACCESS_COARSE_LOCATION	getLatitude
		getLastKnownLocation
		getAltitude
		getLocation
Contacts	READ_CONTACTS	getAccounts
	WRITE_CONTACTS	
	GET_ACCOUNTS	
Camera	CAMERA	-
IMEI	READ_PHONE_STATE	getDeviceId
		getImei
Calendar	READ_CALENDAR	-
	WRITE_CALENDAR	
Audio	RECORD_AUDIO	-
Email	-	getEmailAddress
Phone Number	READ_PHONE_STATE	getLine1Number

Table 4-7:List with Examples of Sinks That Are Used in LPM.

Sinks	Permissions	API Calls
SD card	WRITE_EXTERNAL_STORAGE	getExternalStorageDirectory
	READ_EXTERNAL_STORAGE	
SMS	SEND_SMS	sendTextMessage
	RECEIVE_SMS	sendMultipartTextMessage
	READ_SMS	sendSms
	RECEIVE_WAP_PUSH	
	RECEIVE_MMS	
Sharedprefs	-	getSharedPreferences
HTTPConnection	INTERNET	Connect
	ACCESS_NETWORK_STATE	openConnection
		loadUrl
Bluetooth	BLUETOOTH	checkBluetooth

Moreover, the clusters with similar applications that were created through the descriptions to calculate the functionality score are used to discover the relation between the applications and the predicates, any behavior (predicates) that is not common may be considered as an outlier that may be leaking sensitive information. For instance, if an application requests location and user contacts info but other applications from the

cluster requests only location, in this case the user contacts request info is highlighted as it can be an unnecessary or dangerous behavior and also an overpermission problem.

One-Class Support Vector Machine (OC-SVM) is used to identify the outliers, this kind of machine learning algorithm is used to learn the features of one class of elements and it is commonly used when there exists a lot of examples of one class of instances, in our case the similar apps so everything else will be considered as an outlier.

With the Permissions and API calls as features vector an OC-SVM model is created and trained and can later be used to identify the outliers and retrieve the uncommon predicates that are later shown to the user in a report.

4.6. Final Considerations

Through the evaluation of the architecture it is possible to observe that it has some characteristics that do not exist in previous works, such as the security layer used before the recommendation phase to suggest only apps classified as benign, the calculation of metrics within a context of functionality to create a rank only with apps that have similar goals and the mapping between the apps features and descriptions so users will be able to understand all possible behaviors that the app can take and also check any behavior that is suspicious.

Next Chapter presents an evaluation of the developed prototype through a comparative analysis in different test scenarios with other tools chosen from the related works and with the official store.

CHAPTER

5

Experimental Results

This Chapter presents the results obtained by evaluating the prototype. Quantitative and qualitative experiments were carried out to verify the approach. In quantitative experiments it is verified which classification model is more efficient in the identification of malicious applications and which parameters are best for the recommendation model. In qualitative experiments some malicious applications were chosen to be evaluated inside the prototype and in other frameworks to compare the results. RSPSA [60] and DroidVisor [58] were the frameworks selected, these works were described in Section 2 and were chosen because they have characteristics that are similar to the proposed system. Google Play was also compared to check how the official store makes its recommendations. Finally, an analysis of the features was made to find out the ones that best characterize malicious apps in the dataset used.

5.1. Experimental Data

A machine learning classification model was created to evaluate the prototype, J48 Decision Tree algorithm was executed with its default parameters. In total, 35 sensitive API calls and 60 permissions were defined from the analysis of 1954 applications obtained from [73]. Some selected features have also been used in other works related to malware detection in Android [74], [75], [76]. The training set has 1648 benign apps and 307 malicious apps, representing 47 official store app categories such as sports (43), productivity (49), communication (46), finance (49) and business (62).

Figure 5-1 and Figure 5-2 show some statistical data from the applications that are in the dataset that also are in the Google Play Store. Specifically, Figure 5-1 shows the percent number of applications by category along with the requested permissions average. The chart shows that the dataset has more applications in the Tools, Education and Business categories and the apps from the Communication category request more permissions.

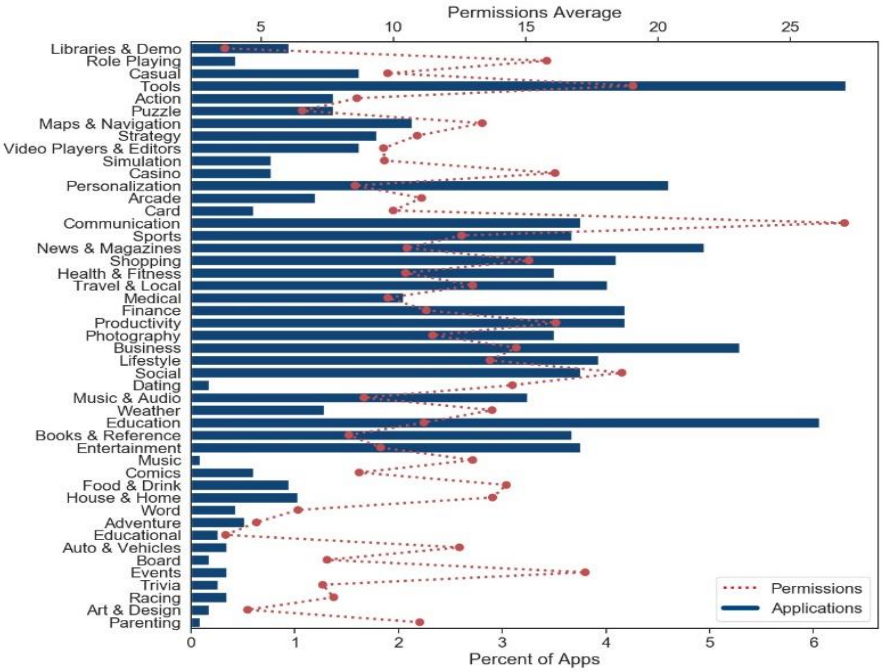

Figure 5-1: Percent of Apps and Average Number of Requested Permissions by Category.

Figure 5-2 shows the percent number of applications by category along with the API method Calls request average. In this chart, the category that most executes API Calls is Weather followed by Travel & Local and House & Home.

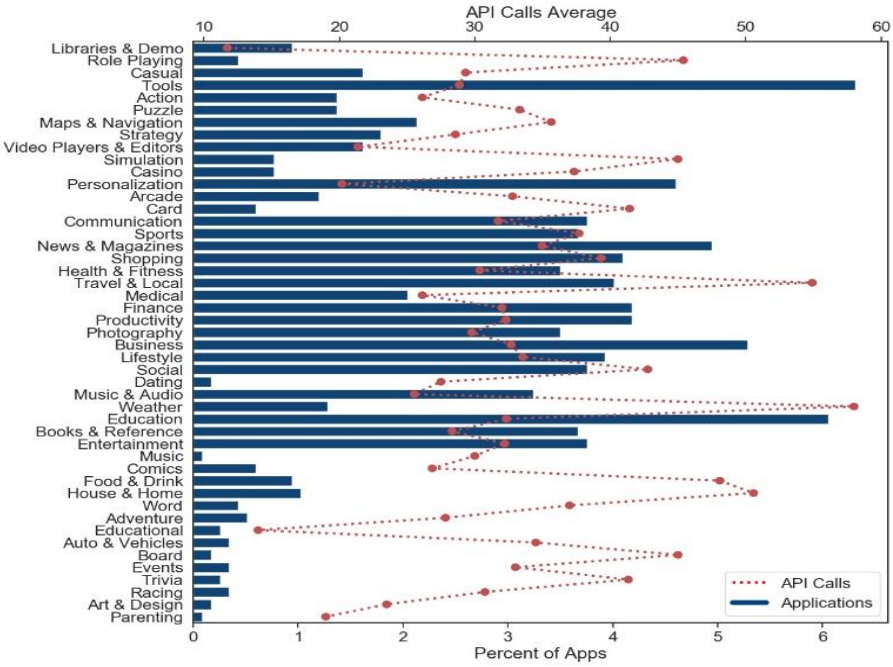

Figure 5-2: Percent of Apps and Average Number of API Calls by Category.

Moreover, 100 apps were crawled from Google Play and added to the database to enrich the suggestions. Finally, the model is always incremented and retrained with the apps that are evaluated. The metrics used for the quantitative experiments were Precision, Recall and F1-Score for classification while coherence score is used to evaluate the recommendation model. Table 5-1 shows the classification model evaluation results with four algorithms.

Table 5-1: Machine Learning Model Evaluation.

Algorithm	Precision	Recall	F1-Score
J48	96,50%	96,50%	96,50%
Logistic	95,60%	95,70%	95,70%
SMO	94,20%	94,40%	94,42%
Naïve Bayes	90,60%	84,60%	84,55%

From the table, Precision is related to the question "of all applications labelled as malware, how many applications actually were malware?" while Recall is related to "of all the applications that are really malware, how many did we label?" and F1-Score combines Recall and Precision to reach a ratio that measures the overall quality of the model. Since we need a model with a good Precision and with a good Recall, J48 was chosen because it has the highest F1-Score.

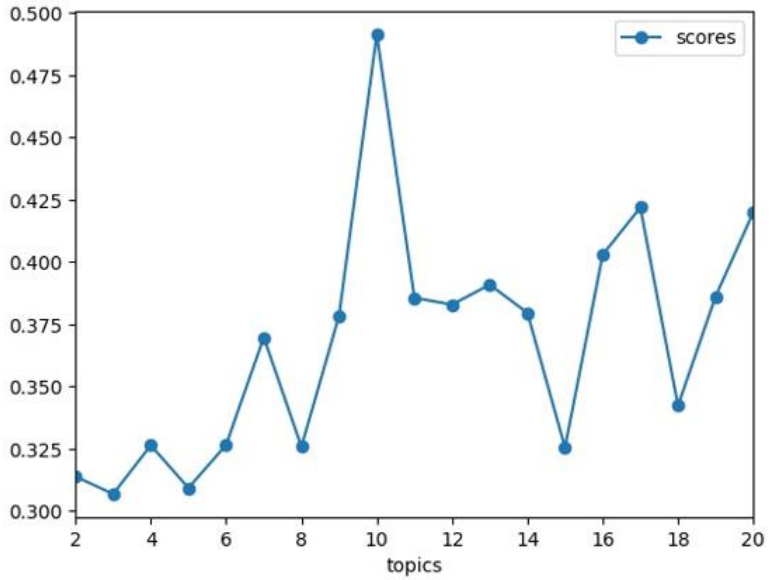

Figure 5-3: Coherence Score Values by Number of Topics.

The recommendation model was developed in Python using libraries such as Pandas and NLTK for preprocessing to remove stop words, punctuation, words with small size, lemmatization and other strategies. Gensim for LDA and pyLDAvis and WordCloud for visualization. Coherence score was chosen because it takes a topic and measures the degree of semantic similarity between the words with the highest score in the topic [77]. These calculations help differentiate between topics that are semantically interpretable and topics that are only artifacts for statistical inference.

The evaluation was made with the list of descriptions from the Play Store communication category, Figure 5-3 shows an average of coherence values. To reach these values a different number of generated topics were used.

Since LDA training is non-deterministic each coherence value was obtained through an average of executions and the best model was the one with 10 topics with an average coherence score value of 0.49. Next section presents the qualitative experiments with the baselines.

Table 5-2: Malicious Apps Selected to Compare RSPSA and the Prototype Created Results.

Name	Category	Possible Behaviors
Spam Guard	Productivity	Steal information from the compromised device.
Cut The Rope Unlock	Puzzle	Install packages, uninstall packages, run packages, check version information, open a URL, download and install additional files.
Deal&Be Millionaire	Trivia	Copy bookmarks, push notifications, shortcuts, identify the last executed command and steal information.
Fish Aquarium Live Lock	Personalization	Steal information from the compromised device.
Voice SMS	Communication	Steal information from the compromised device and sends SMS messages.
Where is My Water?	Puzzle	Steal information from the compromised device.
ClockPlus	Tools	Sends SMS to premium services.
Banco do Brasil	Finance	Shows adds and steals information.
Sberbank	Finance	Requests administrator rights and then make themselves invisible in the list of installed apps to intercept user's personal data, such as SMS.
Opera Mini 6.5	Communication	Steal information from the compromised device.

5.2. Prototype vs RSPSA

In order to compare the created Prototype results with RSPSA, 10 malicious applications from different categories and with different objectives were selected. Table 5-2 lists the apps.

The applications were downloaded from Koodous [78] website and were chosen because they perform tasks that are usually executed by the users in benign apps. For instance, Voice SMS is an application that transforms speech to text and writes SMS messages through voice. ClockPlus is a stopwatch to measure the time of an activity such as games, cooking and education, while Banco do Brasil and Sberbank are bank applications. Some applications are also repackaged versions (modified version with malicious code) of well-known applications such as Opera Mini 6.5, which is a modified version of the Opera web browser, and Cut the Rope, which is a modified version of a famous puzzle game.

Spam Guard was the first malware application evaluated, which is from the productivity category and its goals are described as an application that automatically detects and moves spam emails from the user inbox to the spam folder. However, the application accesses users' sensitive information such as contacts and sends it through SMS messages.

The recommendation strategy used in RSPSA receives a list of applications from the same category (productivity in this case) and calculates user ratings and security scores based on permissions configuration. Then, these results are used in a clustering algorithm to perform the suggestions. Table 5-3 shows the top 3 applications related to Spam Guard after RSPSA evaluation and the top 3 applications suggested from the proposed prototype.

Table 5-3: Spam Guard Results.

Approach	*Application*	*User Rating Score*	*Permission Score*
RSPSA	Xodo PDF Reader & Editor	4.72	7.0
	Business Calendar 2	4.61	28.0
	Password Safe - Secure Password Manager	4.61	4.0
Proposed System	Email Spam Filter	2.60	11.0
	Email - Fast & Secure mail for Gmail Outlook & more	4.60	18.0
	Microsoft Outlook	4.33	16.0

None of the applications from RSPSA results have the same goal as Spam Guard. For instance, Xodo PDF is a PDF reader and editor while Business Calendar 2 is a calendar. Meanwhile, the proposed prototype recommendation strategy discards the malicious app and suggests applications with similar functionalities. The Email Spam Filter application stands out as it is used to control and restrict which emails are added to a user inbox.

Email Spam Filter had a 2.6 user rating score and 11.0 privacy score. A low user rating score shows that the application is poorly accepted by users regarding features related to usability, such as user interface. However, the application has few revisions (193) and can improve this score over time through new user reviews and application updates. The privacy score shows that the app does not require many permissions, with seven normal permissions out of a total of 33 and three dangerous permissions (READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE and READ_EXTERNAL_STORAGE) out of 27.

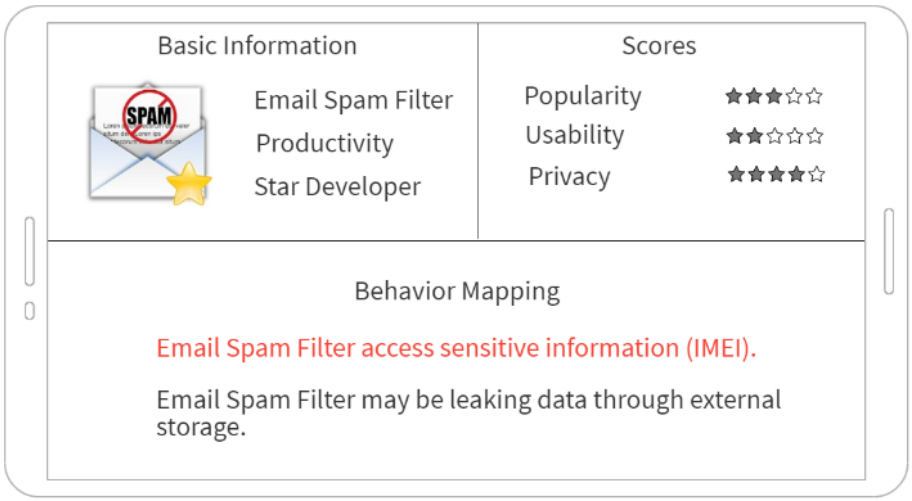

Figure 5-4: Email Spam Filter Results.

The first dangerous permission setting allows the application to retrieve information about the mobile device, such as network information and device number, while the other two permissions configuration allow the reading and writing to external storage. The privacy score value is not normalized because RSPSA has not done any normalization. Thus, the higher the privacy score is, the greater the risk of data leakage is. Figure 5-4 provides a screenshot from the proposed prototype with results referring to Email Spam Filter showing the application name, category, developer, and scores. Regarding the LPM, access to IMEI and writing to external storage behaviors were found and mapped, which

is important as the IMEI access is an uncommon behavior in the in the Email Spam Filter similar apps cluster. This information is grouped and passed on to users so they are aware that the app can access this confidential information and may cause potential data leakage.

Table 5-4 shows the results with the other 9 malicious applications. Banco do Brasil application is a repackaged version of the legitimate bank application with the goals to steal information and show adds. RSPSA top application recommendation was Canadian Mortgage App, while the prototype recommended the real version of Banco do Brasil application. The description provided from the malicious application caused the real and secure version of Banco do Brasil to be suggested because of the functionality score.

Table 5-4: Malicious Applications Evaluation.

Malicious Apps	*Category*	*RSPSA*	*Proposed Prototype*
Cut The Rope	Puzzle	I love Hue	Cut The Rope (Benign)
Banco do Brasil	Finance	Canadian Mortgage App	Banco do Brasil (Benign)
Deal&Be Millionaire	Trivia	Millionaire Trivia: Who Wants To Be a Millionaire?	Millionaire 2019 - Trivia Quis
Fish Aquarium Live Lock	Personalization	New Year 2019 countdown	Fish Live Wallpaper 2018
Voice SMS	Communication	Pleymojis	Write Voice SMS
Where is My Water?	Puzzle	I Love Hue	Where's My Water? 2
ClockPlus	Tools	Post	Multi Timer StopWatch
Sberbank	Finance	Canadian Mortgage App	Sberbank Mobile Bank
Opera Mini 6.5	Communication	Pleymojis	Opera Mini - fast web browser

Deal&Be Millionaire is a game from the Trivia category. It is a malware that steals information and its description says that it is a game to become the next millionaire through intense deals and risky decisions. RSPSA suggested the app Millionaire Trivia: Who Wants To Be a Millionaire? while the prototype suggested Millionaire 2019 - Trivia Quiz. This is the only case that RSPSA suggested an application that has the same goal of the application being evaluated. Since RSPSA gets the applications from a category and calculates the overall user rating and permission scores, it always returns the same top suggested application. For instance, for the Communication category it always returns Pleymojis application, while for the Finance category it always returns Canadian Mortgage App.

On the other hand, the prototype considers the apps descriptions to check its functionalities. Therefore, it returns different results depending on the app being evaluated. For instance, in Table 10, inside the Communication category two different apps were suggested: Write Voice SMS: write SMS by voice for Voice SMS and the real Opera mini browser for Opera Mini 6.5.

5.3. Prototype vs DroidVisor

Another comparison is made with DroidVisor. A tool that claims to recommend secure apps based on requested permission configurations. DroidVisor also uses similarity, usability and popularity as metrics, users can specify weights to the metric that they consider more important, since the focus of this work is on security, this metric received the highest weight.

In this case study, we chose MobonoGram application, a malicious app that was available in 2019 inside Google Play Store and was recently removed. A recent study showed that the average time a malicious application spends inside the Google Play Store is about 51 days and can stay up to 138 days [79]. MobonoGram was developed from RamKal, has a 18M size, requires Android 4.1 and up and its publish date was 22/05/2019. During its time inside the store, there were 100,000 downloads and an average user rating of 3.8. MobonoGram claims to be an unofficial version of Telegram with much more functionalities and other unofficial features. However, when executed MobonoGram runs services on background without user consent and also loads and browses malicious websites [80]. Table 5-5 shows the Top 3 results with no normalization to compare the results.

For DroidVisor results, MobonoGram and Telegram had the same privacy score since MobonoGram is based on Telegram. They almost had the same number of permission configurations (Telegram 47 and MobonoGram 52), with service-related permissions such as BIND_JOB_SERVICE and activity recognition permissions such as ACTIVITY_ RECOGNITION being included among the uncommon permissions between the apps. DroidVisor ended up recommending MobonoGram as the first application (same score as Telegram), even though it is a malicious app. This is because DroidVisor does not classify applications, it only analyzes app permission configurations and assumes that the application with the highest privacy score (GO SMS Pro) is the most dangerous. However,

there are insecure apps that request few permissions configuration as shown in Chapter 1.

Table 5-5: Prototype vs DroidVisor Results with MobonoGram.

Approach	*Application*	*Privacy Score*
DroidVisor	MobonoGram	16.96
	Telegram	16.96
	GO SMS Pro - Messenger, Free Themes, Emoji	23.62
Proposed System	MobonoGram	Discarded
	Telegram	6.59
	JusTalk - Free Video Calls and Fun Video Chat	7.99

Unlike DroidVisor, the proposed prototype discarded MobonoGram because it was detected as a malicious application at the security layer, confirming the importance of the classification machine learning model inside a recommendation system with security and privacy awareness. Lastly, the difference between the privacy scores (Telegram had a privacy score of 16.96 in DroidVisor and 6.59 at the proposed prototype) happened for two reasons: First, DroidVisor permission weights are divided into low, medium and high with the values being 0.33, 0.66 and 1.0, respectively, and the authors do not explain how these values were designated. On the other hand, the proposed system calculates the weights in a more sophisticated way based on the probability of the feature being requested in the apps from the same category, with that the weights are fairer and more varied.

Second, DroidVisor evaluates all the permissions that are being requested from the application, while the proposed approach considers only the permissions that are not common between the apps being recommended with a technique called functionality-based privacy score. This means that apps with the same functionality, i.e., applications that send SMS, will not have permissions related to SMS included in their privacy score formula because the applications really need to request these permissions configuration. Therefore, the prototype privacy score is fairer and more accurate. This is important because the privacy score may change the order of suggestions and recommend different apps in other scenarios.

5.4. Prototype vs Google Play

The last applications comparison is made with Google Play Store since it is the official Android store and most of the users download their applications or get suggestions from it. Table 5-6 shows the related apps that are returned if a user searches for Viber, a famous messaging application, inside the store and also the results with the proposed system.

For Viber, Google Play suggests IMO free video call and IMO beta free call that are applications created from the same developer, as well a Mail.ru that is an email application. Meanwhile, the proposed system suggests Telegram, Messenger and GO SMS Pro - Messenger, Free Themes, Emoji, which are all messaging apps with similar functionalities as Viber.

Table 5-6: Prototype Versus Google Play Results with Viber.

Approach	*Application Recommended*
Google Play	IMO free video call IMO beta free call Mail.ru - Email App
Proposed System	Telegram Messenger GO SMS Pro - Messenger, Free Themes, Emoji

Besides not considering any security aspects, Google Play apparently makes its recommendations by prioritizing the app developers over functionality. An improvement in the store would be adding filters for the users to choose how they want their recommendations.

In addition, Google Play categories could be broken down into more specific categories to prevent apps with different functionalities from falling into the same category. To prove that fact, the topics distribution generated from the communication Google Play category are show in Figure 5-5, each bubble represents a topic and the size of it measures how prevalent the topic is relative to the data.

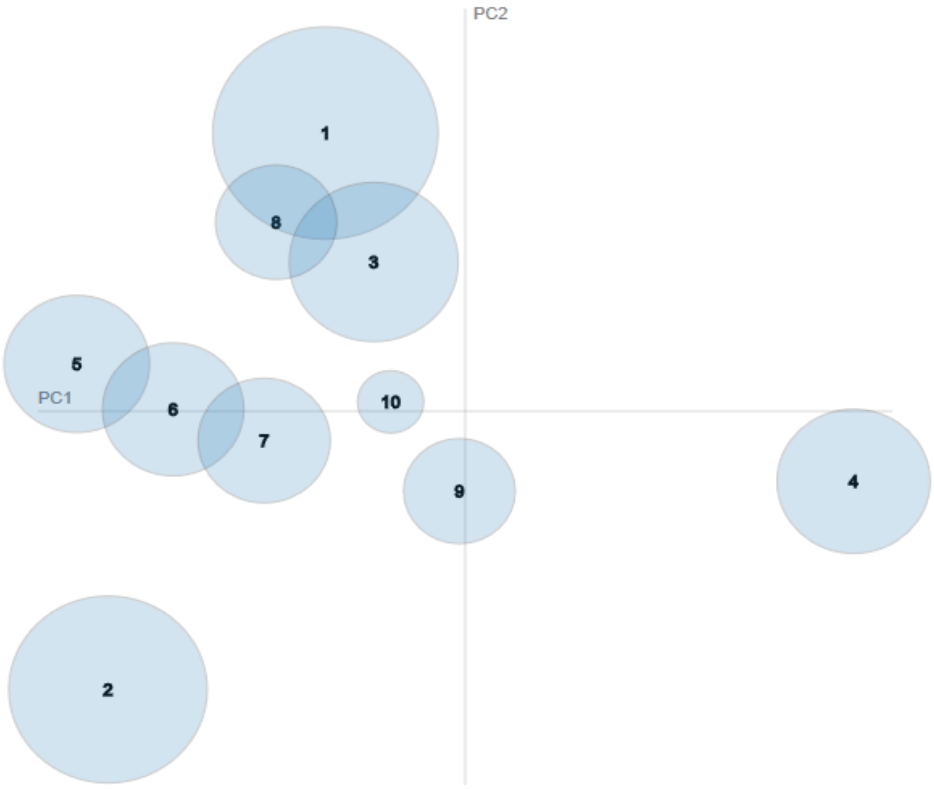

Figure 5-5: Topics Distance Mapping.

There are certain topics such as 2, 4 and 9 that are far away from the position that concentrates most bubbles and could be a new category in Google Play Store while the overlapping bubbles could be analyzed and merged into a category that covers them. Figure 5-6 shows the world clouds with the top 10 most frequent words in topics 5,6,7 and 1,8,3 because these topics are overlapping and share worlds and from topics 4 and 9.

Topic 4 has words such as ‘imap’, ‘account’, ‘email’, ‘client’ and ‘mail’, Topic 1 has ‘phone’, ‘number’, ‘sms’, ‘message’ and ‘text’. While, Topic 6 has words such as ‘blacklist’, ‘dialer’, ‘voip’, ‘identity’, ‘call’ and Topic 9 has ‘encryption’, ‘email’, ‘privacy’, ‘protonmail’ and ‘customer’.

All these topics suggest that the communication category could be broken into other smaller categories that could be more specific. For instance, Topic 1 suggests the creation of a category related to applications that send SMS text messages while Topic 4 suggests the creation of a category with applications related to email services such as Gmail and Outlook.

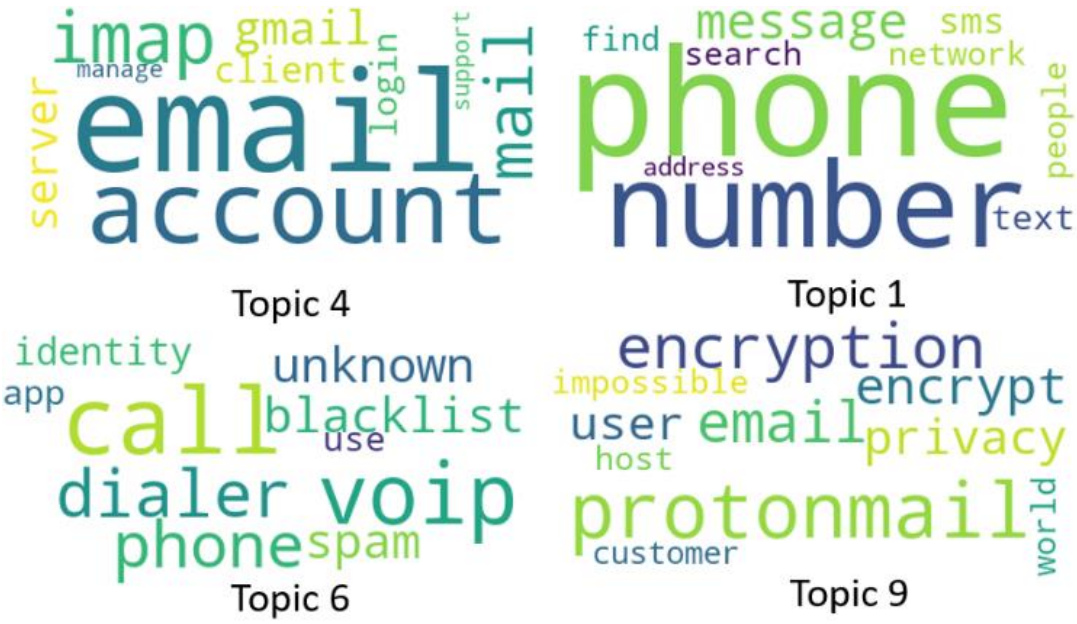

Figure 5-6: Top 10 Most Frequent Words in Some Topics.

5.5. Features Analysis

The features were also analyzed to verify the impact of the permissions and the API call in the detection of malicious applications. Figure 5-7 shows a chart with the top 10 most used features in benign apps, while Figure 5-8 lists the top 10 features from malicious apps.

The experiment shows that 8 of the 10 most used features are method calls for benign applications, while five are permissions and the other five are method calls for malicious applications. This result shows how important it is to use other features, besides permissions, when evaluating mobile applications security.

There are four features from the list of most used features in malicious applications that do not appear in the list of the benign applications features (READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, RECEIVE_BOOT_COMPLETE, getDeviceId).

READ_PHONE_STATE allows the application to retrieve some information from the mobile device such as the mobile device number and network information. WRITE _EXTERNAL_STORAGE allows the app to write data to an external storage such as a sdcard. getDeviceId is a method call that returns the phone IMEI and RECEIVE _BOOT_COMPLETE allows the app to receive a message when the operational system finishes booting.

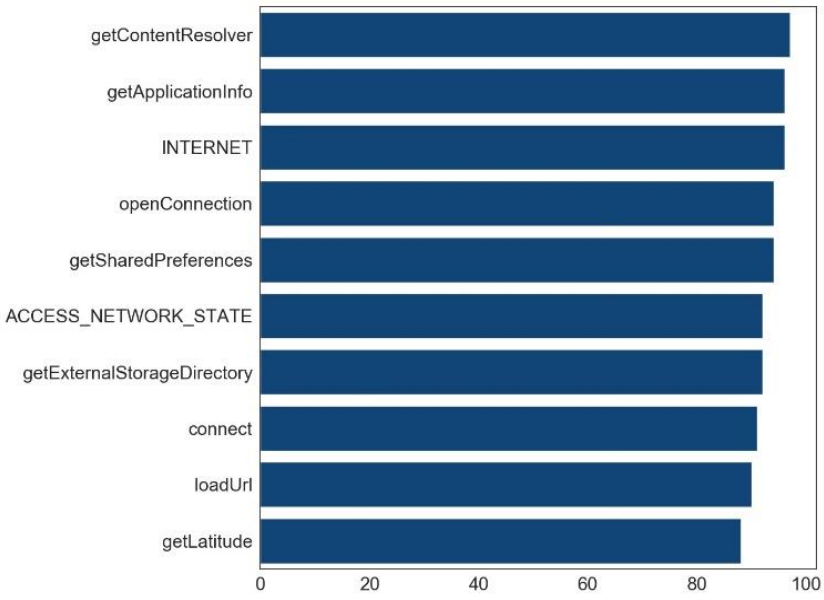

Figure 5-7: Top 10 Most used Features in Benign Apps.

In addition, there are three features that are widely used in malicious applications and almost never appear in benign applications (SEND_SMS, RECEIVE_SMS and getLine1Number) with ratios of 33%, 31% and 30%, respectively. The first two are related to SMS messages sending and are related to malicious applications that send sensitive information to third-party servers and/or subscribe to premium SMS messages services, while getLine1Number is an API method call that returns the mobile device number.

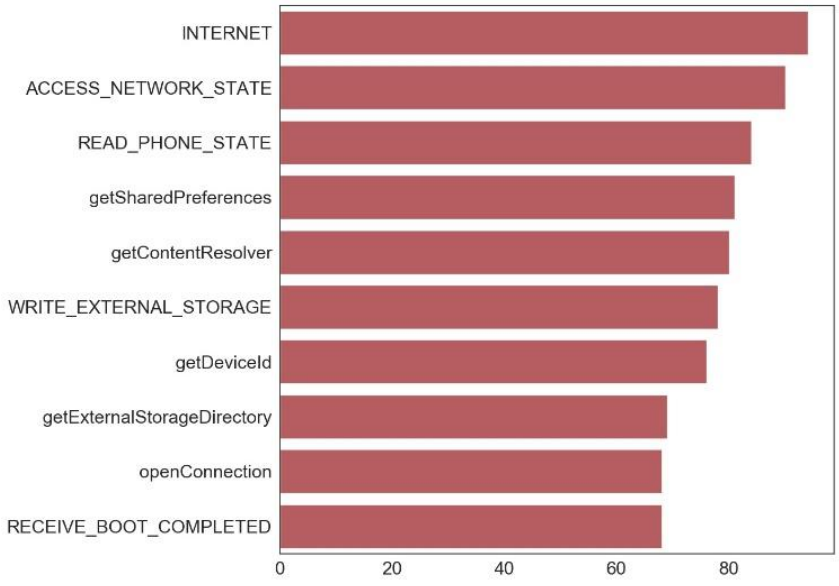

Figure 5-8: Top 10 Most Used Features in Malign Applications.

These features, along with the features from Figure 5.7 that do not appear in Figure 5.8, can be considered good choices for classifying malicious mobile applications due to the high ratio usage difference.

5.6. Final Considerations

The experiments conducted in the Chapter showed that the proposed system added new features that improved the assessment of security and privacy aspects for mobile applications recommendation and achieved a classification model that has 96,5% F1-Score in Android malware classification.

In addition, the system incorporated algorithms and techniques that are used to recommend applications that have the same goal of the application being evaluated in order to allow the users to still execute their tasks avoiding the recommendation of applications that do not satisfy users and also preventing users from abandoning the use of mobile devices because of the lack of trust related to security and/or privacy aspects.

Finally, the final experiments showed that the permissions are features that can be used in the security evaluations of the applications. However, they can be combined with other types of features to increase the quality and accuracy of the detection model.

CHAPTER

6

Conclusions

This thesis presented a system architecture that is divided into three stages (Feature Extraction, Classification and Recommendation). The first stage, called Feature Extraction, was created to obtain the content from a target application, identify the features and create a vector that represents the features that the application uses.

Second stage (Classification) tests the features vector created in a machine learning classification model that was previously trained with a set of features vector from malign and benign applications. The model goal is to classify the target application into malign or benign. The last stage (Recommendation) calculates metrics such as privacy, popularity and usability through the information obtained from the target application inside a functionality context and applies a probabilistic topic model technique to create clusters that arrange the applications in a more fine grained way according to the apps functionality. Finally, all the information from the target app are grouped in a summary that shows users if any sensitive information is being accessed, if there is any overpermission problem, if there is any behavior outside its functionality dictated in its description and app suggestions with the same goal for the user to decide which app is the best for their convenience.

The architecture presented offered contributions to the security and privacy assessment process in a recommendation system and to the suggestion of most suitable applications for mobile devices users. In addition, it proposes a report model that was created through the features and the app description so that technical and non-technical users can understand the behaviors of the application being evaluated and also strike overpermission.

Regarding contributions to security and privacy assessment, existing works use only the apps permissions configuration to evaluate security aspects. Nevertheless, it has already been proven that this approach is not enough to get satisfactory results. To obtain better results this work made a detailed study on Android environment and selected new

features to create a machine learning model that is able to classify mobile Android applications into malign or benign before any recommendation is made.

Concerning the contributions related to the suggestion of most suitable applications, existing related works and the Play Store consider different aspects to do the suggestions such as Play Store category, developer's names and popularity over functionality. This approach is not enough for mobile device users since an app with similar functionalities is necessary. Beyond that, Play Store has categories that are not fine grained enough and have several completely different apps. To obtain better results this work calculates all the metrics inside a functionality-based environment. In relation to the report model, existing works outlines only the applications names, some metrics such as popularity, usability and permissions that is inappropriate because most users do not understand how permissions work. To increase user understanding this thesis presented the Logical Predicate Mapping (LPM).

6.1. Limitations And Future Works

The architecture proposed has some limitations. The first one is related to the APK reverse engineering process, because the feature extraction depends on the success of the code conversion from .DEX to Java. Common know limitations from static analysis can also impact the architecture, such as obfuscation and reflection. Fortunately, in our tests we did not have that kind of problem.

Another limitation is related to the extraction of features from the official Play Store. If the store changes its user interface, removes or adds some information, the system may be impacted. During the prototype development this happened once. The last limitation is related to LDA since it does not work well with short documents. In order to mitigate that small descriptions have been removed in the data preparation phase and also the prototype user interface asks users to enter descriptions with at least 250 characters.

Despite the limitations, the experiments showed good and promising results and pointed to an improvement in security and privacy awareness, app recommendation and user understanding. For future works it is intended to identify other features that can characterize malicious and benign apps, with that the machine learning classification model can be improved and accomplish better results. To achieve this goal other techniques to analyze mobile applications could be used such as dynamic taint tracking to obtain features during runtime and minimize some of the static analysis limitations. Besides that, we aim to crawl more applications from different stores to increase the recommendation database to improve the suggestions and the Logical Predicate Mapping. Finally, we aim to test Natural Language Processing techniques at LPM to check the relation between the apps descriptions and the permissions to improve the detection of outliers.

Acknowledgement

This research, according for in Article 48 of Decree nº 6.008/2006, was funded by Samsung Electronics of Amazonia Ltda, under the terms of Federal Law nº 8.387/1991, through agreement nº 003, signed with ICOMP/UFAM.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Part of the results presented in this paper were obtained through research on a project titled "A System to Detect and Prevent Data Leakage in Android Environment", sponsored by Samsung Electronics of Amazonia Ltda, under the terms of Brazilian federal law No. 8.387/91. (SUFRAMA).

References

[1] Elizabeth Edwards, Joanna Lumsden, Julian Rivas Gonzalo, et al. (2016) "Gamification for health promotion: systematic review of behaviour change techniques in smartphone apps" BMJ Open, vol. 6, pages 1-9.

[2] Statista, "Number of smartphone users worldwide" Available in: https:// www.statista.com/statistics/ 330695/number-of-smartphone-users-worldwide/. [Accessed September 12, 2018].

[3] Statista, "Annual number of mobile apps downloads." Available in: https:// www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed: 06-May-2019].

[4] Kun Xu, Weidong Zhang and Zheng Yan (2018) "A privacy-preserving mobile application recommender system based on trust evaluation" Journal of Computational Science, vol. 26, pp. 87–107.

[5] IDC, "Smartphone Market Share." Available in: https://www.idc.com/ promo /smartphone-market-share/os. [Accessed June 20, 2019].

[6] Symantec, "Internet Security Threat Report" Available in: https://www .symantec .com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf. [Accessed April 10, 2017].

[7] ThreatPost, "Joker Android Malware Snowballs on Google Play" Available in: https://threatpost.com/joker-androids-malware-ramps-volume/151785/. [Accessed January 13, 2020].

[8] Lookout, "New adware found hidden with popular applications in app store" Available in: https://blog.lookout.com/beitaplugin-adware. [Accessed June 5, 2019].

[9] CheckPoint, "ExpensiveWall: A Dangerous packed malware on Google Play" Available in: https://blog.checkpoint.com/2017/09/14/expensivewall-dangerous-packed-malware-google-play-will-hit-wallet/. [Accessed July 28, 2018].

[10] Forbes, “New Google Android Malware Warning Issued To 8 Million Play Store Users” Available in: https://www.forbes.com/sites/kateoflahertyuk/2019/10/24/new-google-android-malware-warning-issued-to-8-million-play-store-users/#51b93baa1235 [Accessed November 28, 2019].

[11] ZDNet, “Malware found preinstalled on some Alcatel smartphones.” Available in: https://www.zdnet.com/article/malware-found-preinstalled-on-some-alcatel-smartphones/. [Accessed January 20, 2019].

[12] Google, “Android Google Play Protect.” Available in: https://www.android.com/play-protect/. [Accessed March 12, 2019].

[13] Soti Central, “Google Play Protect Stopping App Install.” Available in: https://discussions.soti.net/ thread/google-play-protect-is-suddenly-stopping-my-app-install/. [Accessed July 20 2018].

[14] Symantec, “Malicious Apps Persistently Appearing in Google Play.” Available in: https://www.symantec.com/blogs/threat-intelligence/persistent-malicious-apps-google-play. [Accessed February 20 2018].

[15] Lenovo, “Google Play Protect Destroyed Bluetooth.” Available: https://forums.lenovo.com/t5/Moto-G4-Moto-G4-Plus-Moto-G4/Google-Play-Protect-destroyed-Bluetooth-deleted-APP-bluetooth/td-p/3799669. [Accessed October 10 2019].

[16] AV-Test, “Test Antivirus Software for Android.”. Available: https://www.av-test.org/en/antivirus/mobile-devices/. [Accessed September 10 2019].

[17] Nuray Baltaci Akhuseyinoglu and Kamil Akhuseyinoglu (2016) “AntiWare: An Automated Android Malware Detection Tool based on Machine Learning Approach and Official Market Metadata” Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), pages 1-7.

[18] Ignácio Martín, José Alberto Hernández, Alfonso Muñoz and Antonio Guzmán (2018) “Android Malware Characterization Using Metadata and Machine Learning Techniques”, Security and Communication Networks, pages 1-11.

[19] Su Mon Kywe, Yingiju Li, Kunal Petal and Michael Grace (2016) “Attacking Android Smartphone Systems without Permissions”, Conference on Privacy, Security and Trust (PST).

[20] Paloalto, “Cloak and Dagger attack with no permission.”. Available in: https://unit42.paloaltonetworks.com/unit42-android-toast-overlay-attack-cloak-and-dagger-with-no-permissions/. [Accessed in November 10 2017].

[21] Rui Liu, Junbin Liang, Jiannong Cao, Kehuan Zhang, et al. (2016) “ Understa nding Mobile Users ’ Privacy Expectations : A Recommendation-based Method through Crowdsourcing”, IEEE Transactions on Services Computing, vol. 12, pages 304–318.

[22] Haoyu Wang, Jason Hong and Yao Guo (2015) “Using Text Mining to Infer the Purpose of Permission Use in Mobile Apps”, ACM International Joint Conferen ce on Pervasive and Ubiquitous Computing, pages 1107–1118.

[23] Jiayu Wang and Qigeng Chen (2014) “ASPG : Generating Android Semantic Permissions”, International Conference on Computational Science and Engineering, pages 591-598.

[24] Alessandra Gorla, Ilaria Tavecchia, Florian Gross and Andreas Zeller (2014) “Checking App Behavior Against App Descriptions”, International Conference on Software Engineering, pages 1025-1035.

[25] Irina Shklovski, Scott Mainwaring, Halla Skúladóttir and Hóskuldur Borgtho rsson (2014) “Leakiness and Creepiness in App Space : Perceptions of Privacy and Mobile App Use”, Conference on Human Factors in Computing Systems, pages 2347-2356.

[26] Zdnet, “Malicious Android Photography and Gaming Apps Downloaded 8 Million Times from Google Play.” Available in: https://www.zdnet.com/ article/malicious-android-photography-gaming-apps-downloaded-8-million-times-from-google-play/. [Acessed October 24 2019].

[27] Forbes, “New Android Threat: Google Confirms Malicious Apps Removed.” Available in: https://www.forbes.com/sites/zakdoffman/2019/11/06/new-google-android-threat-these-7-malicious-apps-may-be-downloading-malware-onto-your-phone/#4f5bce3275af. [Accessed October 24 2019].

[28] Alessandra Gorla, Ilaria Tavecchia, Florian Gross and Andreas Zeller (2014) “Checking App Behavior Against App Descriptions”, International Conference on Software Engineering, pages 1025-1035.

[29] Google, “Android Open Source Project.” Available in: https://source.android .com [Accessed January 24 2019].

[30] Nisarg Gandhewar and Rahila Sheikh (2010) “Google Android: An Emerging Software Platform For Mobile Devices” International Journal on Computer Science and Engineering, vol. 1, page 6.

[31] Google, “Security for Android Developers.” Available in: https://developer .android.com/topic/security [Accessed March 20 2019].

[32] Google, “Manifest.permission | Android Developers” Available in: http: //developer.android.com/reference/android/Manifest.permission.html. [Accessed March 20 2019].

[33] Google, “Android Security Best Practices” Available in: https://source.android .com/security/best-practices. [Accessed March 20 2019].

[34] Google, “Secure an Android Device” Available in: https://source.android .com/security. [Accessed March 20 2019].

[35] Ryantzj, “Android Application Package Structure” Available in: http://www .ryantzj.com/android-applicationpackage-apk-structure-part-1.html. [Accessed March 20 2019].

[36] Google, “Application Fundamentals” Available in: https://developer.android .com/guide/components/fundamentals?hl=en. [Accessed March 20 2019].

[37] Dan Boxler and Kristen Walcott (2018) “Static Taint Analysis Tools to Detect Information Flows”, International Conference on Software Engineering Research and Practice, pages 46–52.

[38] Edgar Barbosa, “Taint Analysis” Available in: http://web.cs.iastate.edu/~weile /cs513x/2018spring/taintanalysis.pdf. [Accessed February 25 2019].

[39] Miguel Velez, “Taint Analysis lecture” Available in: https://www.cs.cmu.edu/~ckaestne/17313/2018/20181023-taint-analysis.pdf. [Accessed February 25 2019].

[40] CuckooDroid, “CuckooDroid.” Available in: https://github.com/idanr1986/cuckoo-droid. [Accessed February 25 2019].

[41] CuckooDroid, “Guest Machine Architecture - CuckooDroid v1.0 Book.” Available in:https://cuckoo-droid.readthedocs.io/en/latest/installation/guest_android_on_linux/architecture/. [Accessed February 25 2019].

[42] Aurélien Géron, "Hands-On Machine Learning with Scikit-Learn & TensorFlow" , O'Reilly Media, 2017.

[43] David Blei (2012) “Probabilistic Topic Models”, Communications of the ACM, vol. 55, pages 77–84.

[44] David Blei, Andew Ng and Michael Jordan (2003) “Latent Dirichlet Allocation”, Journal of Machine Learning Research, vol. 3, pages 993–1022.

[45] Keith Stevens, Philip Kegelmeyer, David Andrzejewski and David Buttler (2012) “Exploring Topic Coherence over many models and many topics”, Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 952–961.

[46] Medium, “Your Easy Guide to LDA.”. Available in: https://medium.com/@lettier/how-does-lda-work-ill-explain-using-emoji-108abf40fa7d. [Accessed April 10 2019].

[47] Bahman Rashidi, Carol Fung and Tam Vu (2014) “RecDroid: A resource access permission control portal and recommendation service for smartphone users”, ACM MobiCom Workshop on Security and Privacy Aspects of Mobile Environments (SPME), pages 13–17.

[48] Lingfeng Bao, David Lo, Xin Xia and Shanping Li (2016) “What Permissions Should This Android App Request ?”, International Conference on Software Analysis, Testing and Evolution, pages 36–41.

[49] Bahman Rashidi, Carol Fung, Anh Nguyen and Tam Vu (2016) "Android Permission Recommendation using Transitive Bayesian Inference Model" European Symposium on Research in Computer Security, pages 477–497.

[50] Er-Rajy Latifa and My Ahmed (2016) “A New Protection for Android Applications”, International Journal of Interactive Multimedia and Artificial Intelligence, vol. 3, pages 15–19.

[51] Ankur Shukla, Divya Vikash, Bharavi Mishra and Poonam Gera (2017) “Permission Recommender System for Android”, International Conference on Security of Information and Networks, pages 311–314.

[52] Rui Liu, Jiannong Cao, Kehuan Zhang, Wenyu Gao, Junbin Liang and Lei Yang (2016) “When Privacy Meets Usability : Unobtrusive Privacy Permission Recommendation System for Mobile Apps based on Crowdsourcing” IEEE Transactions on Services Computing, vol. 11, pages 864-878.

[53] Quang Do, Ben Martini and Kim Choo (2014) “Enhancing User Privacy on Android Mobile Devices via Permissions Removal”, Hawaii International Conference on System Sciences, pages 6–9.

[54] Toqeer Ali, Yasar Khan, Tamleek Ali, Safiullah Faizullah, Turki Alghamdi and Sajid Anwar (2018) “An Automated Permission Selection Framework for Android Platform”, Journal of Grid Computing, pages 1-15.

[55] Arnaud Oglaza, Romain Laborde, Pascale Zaraté, Abdelmalek Benzekri and François Barrere (2017) “A new approach for managing Android permissions : learning users preferences”, EURASIP Journal of Information Security, vol. 1, page 13.

[56] Tianli Dang, Zheng Yan, Fei Tong, Weidong Zhang and Peng Zhang (2014) “Implementation of a trust-behavior based reputation system for mobile applications”,International Conference on Broadband and Wireless Computing, Communication and Applications, pages 221–228.

[57] Xin Su, Dafang Zhang, Wenjia Li and Wenwei Li (2015) “Android App Recommendation Approach Based on Network Traffic Measurement and Analysis”, IEEE Symposium on Computers and Communication (ISCC), pages 112–118.

[58] Pulkit Rustgi, Carol Fung, Bahman Rashidi and Bridget McInnes (2017) “DroidVisor: An Android secure application recommendation system” IEEE Symposium on Integrated Network and Service Management (IM), pages 1071–1076.

[59] Maria Gómez, Romain Rouvoy, Martin Monperrus, and Lionel Seinturier (2015) “A Recommender System of Buggy App Checkers for App Store Moderators”, ACM International Conference on Mobile Software Engineering and Systems, pages 1–11.

[60] R. C. Jisha, Ram Krishnan and Varun Vikraman (2018) “Mobile Applications Recommendation Based on User Ratings and Permissions” International Conference on Advances in Computing, Communications and Informatics (ICACCI), pages 1000–1005, 2018.

[61] Hengshu Zhu, Hui Xiong, Yong Ge and Enhong Chen (2014) “Mobile App Recommendations with Security and Privacy Awareness” ACM SIGKDD international conference on Knowledge discovery and data mining, pages 951-960.

[62] Bin Liu, Deguang Kong, Lei Cen, Neil Zhengiang Gong, Hongxia Jin and Hui Xiong (2015) “Personalized Mobile App Recommendation : Reconciling App Functionality and User Privacy Preference”, ACM International Conference on Web Search and Data Mining, pages 315–324.

[63] Konglin Zhu, Xiaoman He, Bia Xiang, Lin Zhang and Achille Pattavina (2016) “How Dangerous Are Your Smartphones ? App Usage Recommendation with Privacy Preserving”, Mobile Information Systems, vol. 2016, 10 pages.

[64] ElMouatez Karbab, Mourad Debbabi, Abdelouahid Derhab and Djedjiga Mouheb (2017) “Android Malware Detection using Deep Learning on API Method Sequences”, Cryptography and Security.

[65] AppBrain, “AppBrain.com.” Available in: https://www.appbrain.com/. [Accessed July 18 2018].

[66] SlideME, “SlideME.” Available in: http://slideme.org/. [Accessed July 18 2018].

[67] Samsung, “Galaxy Store.” Available in: https://www.samsung.com/global/galaxy/apps/galaxy-store/. [Accessed July 18 2018].

[68] Zhenlong Yuan, Yongqiang Lu and Yibo Xue (2016) “Droiddetector: android malware characterization and detection using deep learning”, Tsinghua Science and Technology, vol. 21, pages 114–123.

[69] Android, “Requesting Permissions,” Available in: https://developer.android.com/guide/topics/permissions/requesting.html. [Accessed July 30 2019].

[70] Feng Dong, Yanhui Guo, Chengze Li, Guoai Xu and Fang Wei (2016) “ClassifyDroid: Large scale Android applications classification using semi-supervised Multinomial Naive Bayes”, International Conference on Cloud Computing and Intelligence Systems (CCIS), vol. 6, pages 1-5.

[71] Rohit Goyal, Angelo Spognardi, Nicola Dragoni and Marios Argyriou (2016) “SafeDroid: A distributed malware detection service for android”, International Conference on Service-Oriented Computing and Applications (SOCA), pages 59–66.

[72] Shifu Hou, Aaron Saas, Lifei Chen and Yanfang Ye (2017) “Deep4MalDroid: A deep learning framework for android malware detection based on Linux kernel system call graphs, International Conference on Web Intelligence Workshops (WIW), pages 104–111.

[73] Arashi Lashkari, Andi Kadir, Laya Taheri and Ali Ghorbani (2018) “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification”, International Carnahan Conference on Security Technology (ICCST), pages 1–7.

[74] William Enck, Peter Gilbert, Byung Chun, Landon Cox, et al. (2010) “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones”, USENIX conference on Operating systems design and implementation, pages 393-407.

[75] Omar Tripp and Julia Rubin (2014) “A Bayesian Approach to Privacy Enforcement in Smartphones”, USENIX conference on Security Symposium, pages 175–190.

[76] Yuhao Luo, Dawu Gu and Juanru Li (2013) “Toward Active and Efficient Privacy Protection for Android”, International Conference on Information Science and Technology (ICIST), pages 924-929.

[77] Stefan Jansen, "Hands-On Machine Learning for Algorithmic Trading " , O'Reilly Media, 2018.

[78] Koodous, “Koodous.” Available in: https://koodous.com/. [Accessed April 4 2019].

[79] ElevenPath, “ElevenPath CyberSecurity Report.” Available in:https://www.slideshare.net/elevenpaths/201907151600-foxit. [Acessed February 15, 2019].

[80] Symantec, “Unofficial Telegram App Secretly Loads Infinite Malicious Sites.” Available in: https://www.symantec.com/blogs/threat-intelligence/ unofficial-telegram-app-malicious-sites. [Accessed August 8 2019].