Bridging the gap between Privacy by Design and mobile systems by patterns

Karina Sokolova

Thèse
de doctorat
de l'UTT

Karina SOKOLOVA PEREZ

Bridging the Gap between Privacy by Design and Mobile Systems by Patterns

Spécialité :
Ingénierie Sociotechnique des Connaissances, des Réseaux et du Développement Durable

2016TROY0008

Année 2016

utt
université de technologie
Troyes

THESE

pour l'obtention du grade de

DOCTEUR de l'UNIVERSITE DE TECHNOLOGIE DE TROYES

Spécialité : INGENIERIE SOCIOTECHNIQUE DES CONNAISSANCES ET DES RESEAUX

présentée et soutenue par

Karina SOKOLOVA PEREZ

le 27 avril 2016

Bridging the Gap between Privacy by Design and Mobile Systems by Patterns

JURY

M. B. NGUYEN	PROFESSEUR DES UNIVERSITES	Président (Rapporteur)
M. M. CONTI	PROFESSORE ASSOCIATO CONFERMATO	Examinateur
M. D. LE METAYER	DIRECTEUR DE RECHERCHE INRIA	Rapporteur
M. M. LEMERCIER	MAITRE DE CONFERENCES	Directeur de thèse

Personnalités invitées

M. J.-B. BOISSEAU	INGENIEUR
M. A. VERNIER	INGENIEUR

Acknowledgments

I would like to express my special appreciation and thanks to my advisor Dr. Marc Lemercier, who has been an exceptional mentor for me. I would like to thank you for encouraging my research and for allowing me to develop as a research scientist. Your advice on my research, as well as on my career, and your trust in me have been beyond price.

I would like to thank all the team of Eutech SSII and Alpix, who were always available to share their knowledge and professional experience with me and always willing to discuss and exchange opinions. I would especially like to thank my advisor : Jean-Baptiste Boisseau, one of the co-founders of Eutech SSII, who was enthusiastic about my work and research from the first day we met. During the most difficult times during the writing of this thesis, my advisors gave me moral support and freedom I needed to move on.

I would especially like to thank everyone from University of Technology of Troyes who had trust in me me and who helped me to integrate a French university and French society. I am also very grateful to all the professors who gave me the possibility of expressing myself as a lecturer and lecturer assistant.

A special thanks to my family: words cannot express how grateful I am to my mother, and father for all the sacrifices they have made on my behalf. I would also like to thank all of my friends, in particular my dear friend Natalia Sirina who was always there when no one would answer my queries. Finally, I would like to express my appreciation to my beloved husband Charles Perez sustained me through sleepless nights and pushed me towards my goal. I cannot find the words to thank you for your encouragement all through this adventure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

The first mobile phone was invented in 1908 and has since evolved from a simple voice transmission device into a complex smart piece of equipment offering a number of services. The first mobile phones did not contain much data, but with the evolution of mobile networks, data transfer has been made possible; the connection speed has increased rapidly bringing new services and new data along with hardware and data storage capacity improvement. Nowadays, smart-phones can not only call, send short messages and simple multimedia, provide Internet access like their predecessors, but also propose services for every possible users' need with easily installable programs. Mobile applications - a general name for any small mobile device program - propose personal and business services including media management and sharing, browsers, online magazines, file management, games, weather, travel, calendar, financial, banking, healthcare, lifestyle, social applications, and much more. Easy application search, easy installation via mobile application stores and a large choice of services attract users of all ages and professions.

Smart devices are also equipped with an increasing number of sensors: Near Field Communication (NFC), Global Positioning System (GPS), gyroscopes, orientation, thermometers, proximity and light sensors, pedometers, fingerprints, pressure, rotation and others, enabling the smartphone to have precise information about his owner's state and environment at any given time. This huge amount of personal and business data generated by mobile applications and sensors is not only stored on the device, but also transferred to servers: with accessible fast data transmission smartphones being almost always connected.

The data is precise, the quantity increases every day and the data flow is high and hard to control, and that makes smart devices attractive for malicious actors. Two market leaders face the problem differently. Apple limits the attack vector for iOS [1] devices by restricting the access to the data and by carefully verifying and testing each application before it become available on the official application market - AppStore. Apple's policy is strict about bugs, abusive data usage and also security. Android - a mobile operating system owned by Google - is based on a principle of sharing. Android applications have access to nearly all the native applications' data and can make their data accessible that allows creating a larger variety of services, but giving opportunities to malicious actors

[1] Originally iPhone OS. Mobile operating system created by Apple Inc. and used on mobile devices manufactured by Apple Inc.: iPhone, iPod, iPad.

and greater responsibilities to developers.

Moreover, Google verifies applications only for known malware fingerprints, but never checks the application's behaviour for legitimacy. As a result, GooglePlay - the official Android market - contains not only bug-ridden applications, but vulnerable and even abusive ones, that collects massive amounts of the user's data even if it is not related to the proposed service. It is often considered normal for application providers to require personal data in exchange for cheap or even free service, while users do not always agree. Mobile users are becoming extremely concerned about privacy: there has never been so much private data generated, stored and shared as now, in the era of smart devices and social networks [1].

Dr. Ann Cavoukian observed this massive data usage tendency and reactive, rather than proactive security approach by patches. Back in 2001, she proposed 'Privacy by Design' (PbD) principles that would help to build privacy-respectful systems. The main idea of 'Privacy by Design' is to consider privacy as a key principle at the design stage, building privacy-respecting, secure and proactive systems [2]. Although those principles were successfully applied in some projects, the application of 'Privacy by Design' is limited. First, principles are considered vague and do not offer concrete solutions but have to be adapted to each technology. Second, 'Privacy by Design' is sometimes seen as an obstacle to innovation or the massive adoption of some technologies. For example, the German system 'ELENA' followed 'Privacy by Design'. It was based on an electronic signature card to protect privacy and separate the database-stored data from the individual, while keeping the possibility of authentification, but was abandoned due to the limited adoption of such cards. Finally, information nowadays creates value and revenue, therefore why would system designers and, more precisely, mobile application developers apply 'Privacy by Design' if it is not enforced?

Personal data security and the respect of user's privacy haunt users, developers and also lawmakers. In Europe, the European Data Protection Directive (Directive 95/46/EC) was adopted in 1995. Although at the time the directive was not designed to protect users of the rapidly evolving technologies, it includes 'Privacy by Design' principles such as notification, purpose limitation, etc. A draft of new unified General Data Protection Regulation was adopted in 2015. The General Data Protection Regulation aims to supersede the European Directive and to unify, strengthen and enforce data protection throughout Europe making 'Privacy by Design' not only a benefit for users, but a legal obligation for system designers and developers.

Privacy regulation aiming to control personal data use is also in place outside the EU. United States regulation includes the Children's Online Privacy Protection Act (COPPA) [3] and the California Online Privacy Protection Act of 2003 (OPPA) [4]. Canada has the Personal Information Protection and Electronic Documents Act (PIPEDA) [5] concerning privacy. Although those regulations have similarities with European regulation, they are less severe: for example, PIPEDA does not oblige data controllers and data processors to notify data owners before processing the data and the opt-out method, where user is automatically subscribed for services, is a common practice in the US (Europe enforces an opt-in method, where user must explicitly subscribe to a service).

1.2 Problematic

Information systems often neither follow 'Privacy by Design' principles no the European Union regulation principles. The main reason is a gap between those high level princi-

ples and concrete systems and developers [6]. Many reports such as [7] propose recommendations on mobile privacy improvement repeating basic privacy notions (e.g., data minimization, clear notices) but the exact patterns or technical solutions are often missing. Concrete patterns and guidelines should translate 'Privacy by Design' and European regulation principles into developer-understandable language. This gap is also present in mobile systems.

Mobile privacy was discussed in 'Opinion 02/2013 on apps on smart devices' by the Article 29 Data Protection Working Party [8] adopted on 27 February 2013, giving opinions on mobile privacy and some general recommendations were given. The article states that both the Data Protection Directive and the General Data Protection Regulation are applicable to mobile systems and to all user-dedicated applications within EU. The article clearly highlights four main problems related to mobile privacy: lack of transparency, lack of consent, poor security and disregard for purpose limitation.

The goal of this thesis is to propose pattern-oriented solutions to cope with mobile privacy problems identified by [8] and to give mobile systems more Privacy by (re) Design. This thesis focusses on European Union General Data Protection regulation because, first, this regulation is the most up to date and enforces 'Privacy by Design' principles. Second, the European regulation is more severe and widespread, than Canadian or US laws. This thesis work proposes only client-side mobile solutions (mobile application-oriented). The server side models, software and implementations as well as client-server communication are out of scope of this work.

There are a number of reasons for choosing the Android system for case studies. First, Android is one of the leaders of the modern mobile market. Second, Android is an open system: on one hand, it can be accessed, analyzed and modified freely providing more information for research purposes, on the other hand, the Android openness attracts more malicious actors than other systems. Finally, a number of state-of-the art works have shown that the Android system is weak. We note that, in spite of the fact Android is used for the proof of the concept in this thesis, the proposed methodologies are general and can be applied to any future systems.

1.3 Contributions

Four mobile system problems identified in [8] are the core of the thesis problematic. The main research question to be answered in the dissertation is 'Can we propose developer- and user-oriented patterns to give mobile systems more 'Privacy by Design' and reduce the lack of transparency, lack of consent, poor security and disregard for purpose limitation problems?'

This thesis investigates two axes: (1) the architecture of mobile applications and (2) mobile permission systems.

The following gaps were identified from the state-of-the-art study:

1. Android's poor code quality: lack of unified architecture (axe 1)
2. Poor mobile permission systems and an absence of user-friendly security notifications on Google Play (axe 2)

Based on the identified gaps, this dissertation investigates following research questions:

- What architectural design pattern is suitable for Android application development?

Poor code can be a major source of security faults. Enterprises, such as EUTECH SSII[2], dealing with iOS and Android application development reported that Android application developers have more difficulties than iOS developers not only with security aspects, but also with implementating simple functionalities and clear code. This can reduce the quality of an application and raise costs. Unlike Apple, Google does not enforce any architecture or development guidelines for Android, and developers code according to their experience and knowledge which is often incomplete. Chapter 4 presents an architectural pattern proposed during the thesis and specially adapted for Android: Android Passive MVC. This pattern permits even a novice developer to have a good quality code. Following a concrete pattern developers save time and can focus on system privacy and security, instead of trying to organize a readable and maintainable code.

- Can a user-friendly privacy indicator be generated using application-related data?

Mobile markets propose similar information about each application available on the market to help users choose the most suitable application. This information includes name, description, note, comments, screenshots, icons and, sometimes, privacy policy. Android also includes the list of required access to some sensitive interfaces or data - permissions. State-of-the-art studies showed users cannot evaluate the security or privacy level of mobile applications with the proposed information and need a simpler indicator provided systematically. Chapter 5 presents a risk warning indicator obtained following an analysis of permission usage of Android applications from different categories. The pattern identifies normal permission requests by category and penalizes permissions that are not representative for a particular category. The permission score built on those patterns permits a simple security/privacy level of applications to be determined and leverages the warnings for abnormal applications.

- Can currently used permission systems be improved to cope with European Union law and Privacy by Design?

State-of-the-art studies showed current permissions systems are not adapted to resolving the four problems outlined in [8], but, suitably modeled, permissions system should be able to work those problems out. Chapter 6 presents a novel permission system model including 'Privacy by Design' and European Union law notions such as transparency, purpose limitation and control.

1.4 Document organization

This document is organized as follows.

Part I is dedicated to the context and a study of the state-of-the-art related to the propositions of this dissertation. Chapter 2 presents the current state of Android applications development quality and known architectural design patterns. Chapter 3 presents permission systems and related studies.

Part II presents the methodology and the results of this dissertation in three chapters. Chapter 4 explains the proposed architectural pattern for Android developers. Chapter 5 details the Android permission usage analysis, category patterns construction and its'

[2]This thesis is a project of program CIFRE between EUTECH SSII and University of Technology of Troyes

application to Android application classification and risk warning. Chapter 6 presents the novel permissions system model for mobile devices.

This document ends with the conclusion, a list of related publications and an appendix.

Part I

State-of-the-art

Chapter 2

Privacy, Privacy by Design and European regulation

Abstract

In European Union privacy is recognized as a fundamental human right. This chapter proposes necessary privacy-related definitions and briefly explains legal principles included into the Directive 95/46/EC and European Data Protection Regulation. We also present Privacy by Design principles and make parallels with the Directive and similar principles and concepts known in computer science and computer security fields. We conclude that Privacy by Design is now become a legal obligation in European Union and not simply a benefit. System developers, including mobile developers, must integrate Privacy by Design into their design and development process and need privacy empowering options to be able to understand and to follow Privacy by Design.

2.1 Privacy in European Union

The right for privacy is currently protected in many countries. United States regulation includes the Children's Online Privacy Protection Act (COPPA) [3] and the California Online Privacy Protection Act of 2003 (OPPA) [4]. Canada has the Personal Information Protection and Electronic Documents Act (PIPEDA) [5] concerning privacy. In European Union privacy was originally protected by European Data Protection Directive 95/46/EC [9] that was adopted in 1995 and is dedicated «to protect the fundamental rights and freedoms of natural persons and in particular their right to privacy, with regard to the processing of personal data». To homogenize the privacy protection law over the European Union it was decided to supersede the Directive by the European Data Protection Regulation [10] that will be applicable in European Union without a need for national implementing legislation. On 25 January 2012, the European Commission unveiled a draft for European Data Protection Regulation enforcing the same principles all over the European Union [11]. The regulation enforcement should begin in December 2017.

The personal data is defined in the Directive 95/46/EC as follows: «Personal data shall mean any information relating to an identified or identifiable natural person ("data subject"); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity». Additionally to personal data, the Directive defines particular type of personal data named sensitive. The «data revealing racial or ethic origin, political opinions, religion or other beliefs and health or sexual life», are considered sensitive by the Directive and require additional protection, because such data, if revealed, could «pose a risk to the data subject» .

Any processing (accessing, storing, modifying, transferring, etc.) of personal data is protected by the EU law. A controller is defined by the Directive as a natural or legal person or an authority who «alone or jointly with others determines the purposes and means of the processing of personal data». Processor is someone who «processes personal data on behalf of a controller». Controller and processor have a legal responsibility of being compliant with the law.

The personal data cannot be processed without the consent or the data subject. Data Protection Directive define consent as «any freely given specific and informed indication of the data subject's wishes». The consent should be rather given explicitly (required for sensitive data) or in a way that leave no doubt about the fact the person agrees with the data processing. For consent to be valid, data subject:

- must have been under no pressure and must have a real choice without any «negative consequences if he/she does not consent»
- must have been informed about the data usage it's purpose and consequences with adapted language
- must have the possibility to withdraw consent (even if this right is not directly defined by the Data Protection Directive, it is widely presumed)

Directive 95/46/EC defines five principles for private data processing: lawful processing; purpose specification and limitation; data quality; fair processing; accountability [12]. Any system processing personal data should integrate those principles to be compliant with the European law.

- Lawful processing means the data must be processed «in accordance with the law»; if it «pursues a legitimate purpose»; and «is necessary in a democratic society in order to achieve the legitimate purpose».

- Purpose specification and limitation principle states that the purpose should be clearly defined and visible before any data processing starts. Personal data cannot be used beyond the defined purpose.

- According to the data quality principle, data must be «adequate, relevant and not excessive in relation to the purpose for which they are collected and/or further processed» (the data relevancy principle), «accurate and up to date» «in the content of the purpose» (the data accuracy principle) and must be deleted, anonymized[1] or pseudonymised[2] as soon as the goal is achieved (principle of limited retention).

- The fair processing principle state the data processing must be transparent: data subject should clearly understand what of his private data is used, how, who is using this data and for what purpose. System should ask user's valid consent. Finally, user should have a free access to the collected data.

- Accountability principle state controller must ensure the security of the personal data and must be able to demonstrate the compliance of the data processing with the data protection principles.

Principles defined by the Directive are based on the Fair Information Practice Principles proposed in 1973 by a U.S. government advisory committee and adopted by some U.S. states and some countries. Fair Information Practice Principles includes eight principles: purpose specification, use limitation grouped into the purpose specification and limitation principle by the Directive; data quality and collection limitation that become data quality principle; openness and individual participation grouped into fair processing principle; security safeguards, and accountability are included in the accountability.

2.2 Privacy by Design

«Privacy by Design» was introduced by Dr. Ann Cavoukian in 2012. She claims any system must integrate privacy principles and must do it at the software design stage. Privacy by Design includes seven key principles [2]:

1. Proactive, not reactive

Privacy should be an important part of the system conception: privacy violation possibilities should be identified during the design part of the system and the protection should be anticipated. Effective privacy policy should be defined for each system using personal data.

2. Privacy as the Default Settings

All systems must be made taking the privacy principles as the basic and the most important. Principles as data minimization, unlinkability, data aggregation, use limitation and purpose specification should be applied.

[1] The anonymized data is aggregated and do not contain any personal information and can not be linked any more to any person.

[2] In pseudonymised data personal identifiers are replaced with pseudonymes most often by encryption. Pseudonymised data is considered personal as the data can be relinked to the natural person when the decryption key is known.

- Data minimization refers to the concept the require to minimize the personal data collection, storage, treatment, access, identifiability and disclosure to a strict necessary to perform a task. This principle is implicitly included into the data quality principle of the Directive 95/46/EC. This principle joins the principle of least privilege of information security stating that any system unit or user should have the minimum privileges necessary to achieve the goal. The minimization principle reduce the possible damage could be done by a malicious actor and possibilities of data or system misuse.

- Unlinkability refers to the concept making impossible to determine that two items or users are related to each other and represent the same person. This principle is implicitly included into the data quality principle of the Directive 95/46/EC. This privacy principle is similar to the privilege separation principle in software development and computer security. Privilege separation states that the software should be divided into parts with limited privileges each that would permit to reduce the damage from possible attack.

- Data aggregation in privacy refers to a summarizing of the initial data into a certain statistic eliminating all information that may directly identify the person. This principle joins the anonymity principles such a k-anonymity, l-diversity, t-closeness and differential privacy and also joins the data quality principle of the Directive 95/46/EC.

- According to purpose specification principle (one of the Fair Information Practice Principles), the collection and any usage of personal data must be limited to a defined and limited purpose. The purpose should be specified before any usage starts.

- Use limitation is also one of the Fair Information Practice Principles that states the data cannot be used or disclosed beyond the predefined purpose without the user's consent.

3. Privacy embedded into design

Privacy features specific to the system must be identified and embedded from the conception. Talking about the computer systems and security, this principle is related to the idea of software design and design patterns. Software design is the process done before the actual programming. During this phase developer analyses requirement for the future software, defines needs, solves upcoming problems and plans the software and development process. Software design covers algorithm design, architecture as well as security and code quality questions. Design patterns define reusable solutions for common problems and allows gaining in development time and software quality. Software developers dispose high number of predefined patterns for the design and development process, such as creational, structural, behavioral and architectural patterns, but also security patterns and user interface patterns. A few privacy patterns and architectures also exist. Based on the data processing principles, the author of [13] proposes privacy design strategies to be applied during the system design stage: minimise, hide, separate, aggregate, inform, control, enforce demonstrate. The authors of [14, 15] propose a set of privacy design patterns related to some of the proposed strategies. The summary of the privacy preserving techniques and pattern can be found in [16].

4. Full functionality

This principle consists of the integration of privacy keeping the full functionality without sacrificing neither security, nor privacy. The security of the systems must not be based on privacy: privacy-respectful security solutions must be found.

5. End-to-end security

The proper security mechanisms should be applied to ensure the data safety though the full data lifecycle. This idea implies the computer security Defense in Depth concept (or Castle Approach) where system should have multiple layers of security and certain redundancy in security control all over the system and it's lifecycle. This way, if one security measure fails, the next one still protects the system and allows to gain time to be able detect and respond to an attack.

6. Visibility and Transparency

The service and data usage should be clear for the user: he should understand what data is used for what purpose and should be able to control his private information. This principle is clearly referred in the Directive 95/46/EC by the fair processing principle, the purpose specification and limitation principle and the definition of consent. In information technology, this principle gives particular importance to the user interface, as the user should understand clearly what happens with his data.

7. Respect for the user

System should propose a number of privacy-empowering options for the user. This «Privacy by Design» principle reflects the «Privacy by Default» principle: default parameters of the system should respect the privacy and the user should change them himself to be more exposed. By default the user privacy should be protected without any additional action from the user. In computer security, this principle joins the principle of «Security by default» that states that the default security of the system must provide the maximum level of security even if the system becomes less user-friendly. User should make explicit actions to reduce the security level. This principle also rise the importance of the user interface. The user interface design as well as privacy and security-related notifications must be clear to not misguide the user.

2.3 Conclusion

This chapter provided an overview of privacy law in European Union and linked the legal principles with Privacy by Design principles and some computer system and computer security concepts. Most of Privacy by Design principles are integrated into the Directive 95/46/EC and to the European Data Protection Regulation but some principles go beyond legal obligations such as «proactivity» and «respect for the user» even if they are implicit for the Directive. One can see that Privacy by Design become not only a benefit for the final users, but also a legal obligation in European Union.

Both the Directive 95/46/EC and the European Data Protection Regulation are applicable to mobile systems and mobile applications [8]. This thesis investigate existing mobile systems regarding Privacy by Design and propose a number of privacy and security empowering options covering all seven «Privacy by Design» principles.

Chapter 3

Architectural design patterns and mobile development

Abstract

The code quality is important for fast development, group work, performance, project evolution and even security. Developers, even using different programming languages, meet similar problems over and over. To save time and ensure quality, the predefined solutions called «design patterns» are often used. This chapter explains and compares the state of the art of the architectural design patterns such as model-view-controller, model-view-presenter, presentation-abstraction-control and hierarchical-model-view-controller from the state of the art. The background on Android development and iOS architecture is also given as well as the state-of-the-art of the architectures and design patterns related to mobile development.

3.1 Introduction

The mobile market has grown rapidly in recent years. Many companies propose their services with mobile applications, and the number of mobile applications available to users increases every day. Compared to computer programs, mobile applications often have limited functionalities, shorter shelf life and lower price. New applications must be developed fast to be cost-effective and updated often to keep users interested. The quality of the application should not be neglected, as mobile users are very pernickety and competition is stiff. Moreover, the code quality directly affects the security of the software.

In spite of the fact that most programs and applications seem different, developers meet similar problems to be solved over and over again. Instead of reinventing a solution, the faster way to produce good quality code is to apply already provided solutions, which are often called «Design patterns». Design patterns were first described by the «Gang of Four» in 1994 [17] and remain highly usable in different programming languages. Nowadays many types of design patterns are available to developers: structural, behavioral, security oriented, architectural, etc. Architecture choice is important to ensure quality for mobile applications: mobile applications, as well as other systems, can be complex and evolve over time.

Suitable predefined architecture can improve the development time and application quality by liberating developers from some of the technical choices. It also improves the three non-functional requirements of software structural quality: extensibility, maintainability and performance. A defined architecture could additionally reduce the complexity of the code, simplify the documentation and facilitate collaboration work [18].

This chapter provides a state-of-the-art of architectural design patterns in the order historical appearance order and also gives an overview of iOS and Android development components.

3.2 Architectural design patterns

In order to apply suitable architecture to Android development, the existing architecture should be analyzed. This chapter presents the existing architectural design patterns in order of appearance in the literature.

3.2.1 Model-View-Controller (MVC)

Presented in 1978, Model-View-Controller is the oldest design pattern and has been successfully applied to many systems since its creation [19, 20, 21].

The goal of this model is to separate business logic from presentation logic. The business logic modifications should not affect the presentation logic and vice versa [19]. MVC consists of three main components: Model, View and Controller. The Model represents data to be displayed on the screen. More generally, Model is a Domain model that contains the business logic, data to be manipulated and data access objects. The View is a visual component on the screen, such as a button. The Controller handles events from user actions and communicates with the Model. The Controller also communicates with the View directly if the Model does not need to be changed (e.g., scrolling action). The View and the Controller depend on the Model, but the Model is completely independent. The

design pattern states that all Views should have a single Controller, but one Controller can be shared by several Views.

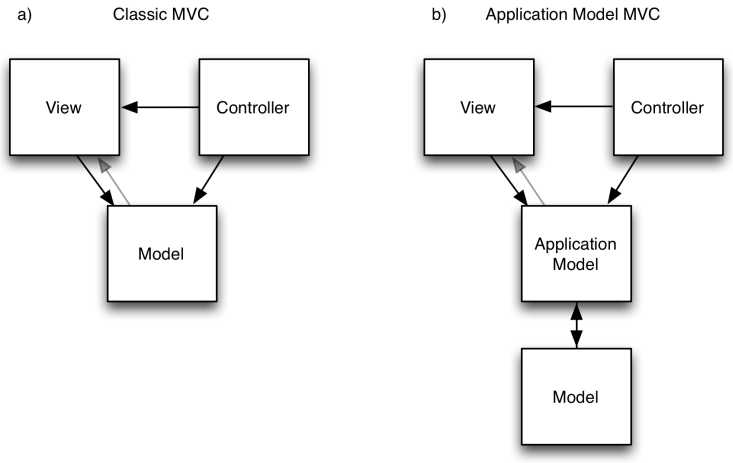

Figure 3.2.1: a) Classic MVC, b) Application Model MVC

The scheme of Classic MVC and Application Model MVC is shown in Figure 3.2.1. The Classic MVC is shown on the left (a) and the AM-MVC is shown on the right (b). The scheme of Passive Model MVC is shown in Figure 3.2.2.

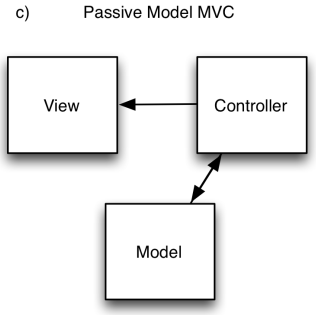

Figure 3.2.2: Passive Model MVC

In Classic MVC and Passive Model MVC, Controller handles events and communicates directly with a Model that is indicated by a black arrow. In the Classic MVC the Model processes data and notifies the View. The View handles messages from the Model and updates the screen using the data received from the Model. This behaviour is implemented using the Observer pattern (grey arrow in Figure 3.2.1). Conversely, the communication between the Model and the View in Passive Model MVC is done exclusively via the Controller. The Model notifies the Controller which then notifies View and finally the View makes changes on the screen [22].

The AM-MVC is an improved Classic MVC with an additional component. The Application Model component was added for the presentation logic (e.g., change the screen colour if the value is greater than 4) that was often previously added to View or Controller previously and makes a bridge between the Model and the View-Controller couples.

3.2.2 Presentation-Abstraction-Control (PAC)

The PAC architecture was introduced in 1987 [23]. This architecture aims to improve the modularity of the system which is limited with MVC. PAC proposes that the system functionalities be decomposed into hierarchically organised cooperating agents each responsible for a particular task. Each agent manages part of the user interface and maintains its data and state. Some agents could also exist without any particular user interface but coordinating other agents. The system can be extended by additional agents, the modification of one agent should not affect other agents.

Each agent of the PAC system consists of three components: Presentation, Abstraction and Control. Presentation component contains the presentation logic. Abstraction component contains the functionality of the agent and the data it maintains. Control component links the Presentation and the Control acting as an adapter and allows communication between agents. One can see that the PAC agent is organised in the same way as Passive MVC with the difference that the user events are intercepted by the Presentation component [24]. Figure 3.2.3 depicts the architecture.

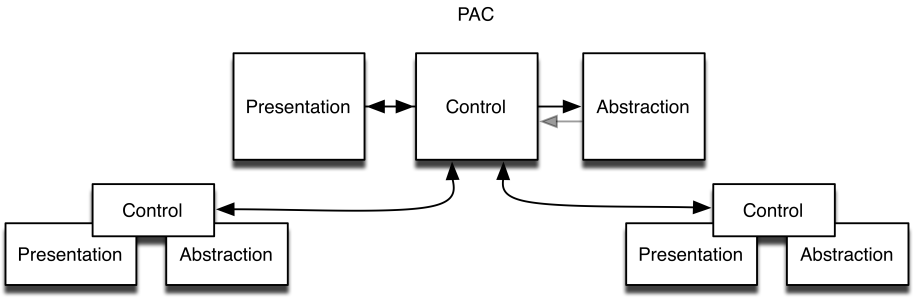

Figure 3.2.3: PAC architecture

Agents are organised in a hierarchy where lower level agents depend on their parents. High-level agents contain the core functionalities, manage the database and main interface. Low-level agents maintain particular functionalities, particular interfaces, the information about the interface and intercept actions from users. Lower level agents could, for example, manage different sensors. Intermediate-level agents combine, maintain and coordinate low-level agents.

The actions intercepted by the low-level agents can be redirected to the upper agents to access their functionalities, the outgoing events such as an error event is also transferred to the particular 'error manager' agent via parentalControl components. The changes in high-level agent's data are also transferred to collaborators agents.

This architecture allows a very modular system with communicating agents to be constructed but the system can become very complex with a rapidly growing number of agents. The organisation or communication between agents can also become complex.

3.2.3 Model-View-Presenter (MVP)

The Model-View-Presenter was introduced in 1996 as an MVC adaptation for the modern needs of event-driven systems [25]. The model consists of three components: Model, View and Presenter. In this model, the View represents a full screen and it handles events from the user actions. The Presenter is responsible for the presentation logic. The Model is a Domain model.

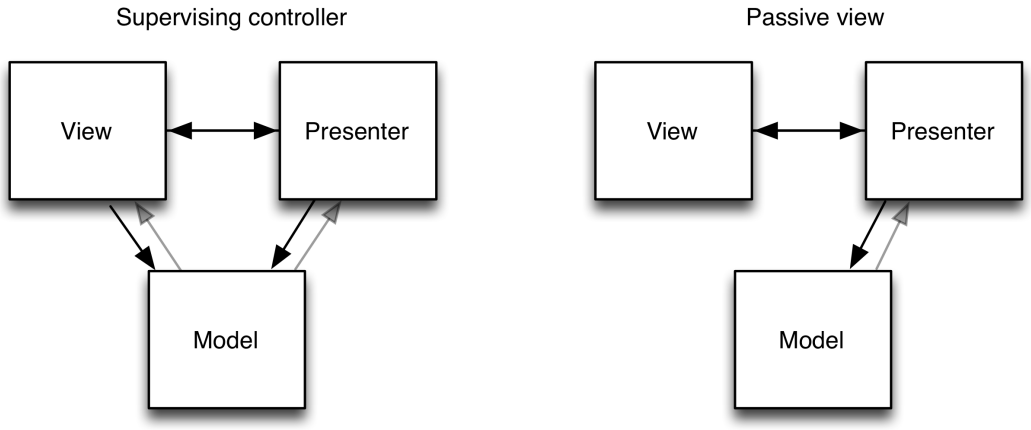

Figure 3.2.4: Supervising controller and Passive view

There are two types of MVP: Supervising controller and Passive view. Both models are shown in Figure 3.2.4. The Supervising controller uses the Observer-Observable pattern for the communication between Model and View. The View can interact directly with the Model to save the data if there is no change to be made on the screen. Otherwise, the communication between the View and the Model is made via the Presenter. Interaction between View and Model of the Passive View MVP is done exclusively via the Presenter [25].

3.2.4 Hierarchical-Model-View-Controller (HMVC)

The Hierarchical-Model-View-Controller was first introduced in 2000 and is similar to PAC architecture. HMVC is presented as a Classic MVC adaptation for Java programming [26]. This model takes into account the hierarchical nature of Java graphical interface components: the main window frame contains panes that contain components. The authors propose creating layered architecture for the screen with Classic MVC triads for each layer communicating with each other by Controllers. The HMVC model is shown in Figure 3.2.5.

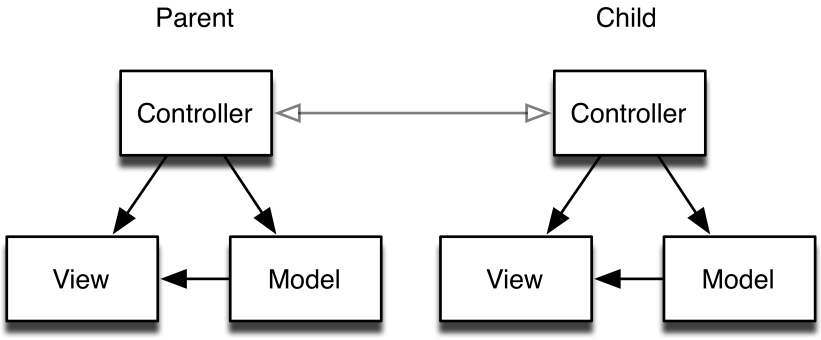

Figure 3.2.5: Hierarchical-Model-View-Controller

Thereby the child Controller intercepts methods from its View. If the View of the upper hierarchy (parent View) needs to be changed, the child component informs the parent Controller, which makes the changes. The communication between layers is made

exclusively via Controllers. Unlike PAC, the Controllers of HMVC have direct access to the Model and to core components without interacting with the high-level triad.

3.3 iOS system and architecture

The logic of iOS applications is similar to Android applications, therefore the iOS experience should be taking into account while choosing suitable architecture for Android.

iOS is an OS X based system adapted to mobile devices. It only runs on mobile devices manufactured by Apple: iPod, iPhone and iPad. iOS developers use specific languages called Objective-C and Swift to create mobile applications.

The base architecture for mobile iOS applications is an adapted Passive MVC. Like the original Passive MVC, the iOS architecture is based on three components: View, Model and Controller. Models and Views are independent and communicate with each other only via Controllers. The communication between Controllers and Model is organized via an Observer-Observable pattern.

Views and Models are highly reusable. Multiple Views are already provided by Apple: SplitView, TableView, ImageView, PageView, CellView, WebView, MapView, TextView, ButtonView, etc. Controllers are less reusable, they link Views with Models, set up the Views (contain presentation logic), and intercept actions made on View to call methods from the Model. Many controllers are already predefined in iOS: ViewController, SplitViewController, TableViewsController, etc.

There is a main controller for each screen or a group of screens exists in iOS applications. For example TabBar represents a menu, and there are as many screens as tabs in this menu. All screens are managed by the same controller - TabBarController. Each screen can embed other Views that can have a corresponding Controller or can be managed by the parent Controller.

3.4 Android system and architecture

Android is a Linux-based open source operating system designed for mobile devices. Android was first presented by Google in 2007 and in spite of huge competition from Apple has been the leading smartphone platform since 2010. Android is widely used on smartphones, tablets, smart watches and even smart TV of different suppliers. Google continues to work on the system systematically integrating new features and correcting bugs. Many manufacturers of smartphones and tablets adopted this open-source solution; the National Security Agency (NSA) and National Aeronautics and Space Administration (NASA) also chose Android for their projects.

Android applications are available for users via the market store - GooglePlay - that in 2015 contained about 1 860 000 applications. GooglePlay is a distribution system that does not include any application verification or certification: the Google Play contract states that the developer is fully responsible for the published application. Although, Google propose many guidelines and tutorials, including security and privacy suggestions, to help developers produce high quality application, Google Play does not control any application nor its behavior. Applications are self-signed by developers and become available on the market a short time after the publication.

Android applications can be also distributed via alternative markets as well as via emails or the web; users need to allow the non Google Play application installation in

parameters.

Android applications are mainly written in Java using the Android SDK [27]. The code is compiled to be executed on the Dalvic virtual machine on a smartphone. Additionally, developers can use the Native Development Kit (NDK) to add a C or C++ written code referred to as native. NDK allows more advanced features and better performance, however, the complexity of the code increases with the quantity of native code [28] – hence Google suggest minimizing the use of this kit.

Google do not impose any particular architecture on developers, but proposes different types of components for different needs. An exhaustive description of Android development environment and modules can be found in [29]. Principal Android components are briefly explained below.

Four principal components of Android SDK are used in Android application development: Activity, Service, Content provider and Broadcast receiver. Developers use predefined extendable classes to implement those components.

Activity is a main mandatory component of Android applications created when the application is opened. The simplest Android application can contain only the class implementing the Activity. Activity is also the entry point to the application: to start the application the system must launch the Activity component. Applications can make the Activity public to share the proposed functionality.

There can be several Activities in the application but only one is active at a time. The Activities history is saved: the system automatically maintains the stack of Activities and opens the previous Activity with its last state when the button 'back' is pressed. The oldest Activities are deleted from the stack for other memory usage.

The Service works on the background of an application permitting the execution of long tasks (e.g., file download) without freezing the screen. When the application is closed, unlike Activity, the work of the Service is not interrupted. Service can communicate directly with the Activity it is attached to.

The Content provider component gives access to the local data stored in SQLite databases. Content provider is aimed to be used for the data sharing between applications but can also be used internally.

The Broadcast receiver is a messaging system that enables communication inside the application and between multiple Android applications installed on the phone.

In 2010, Google introduced a new component into the Android systems called Fragment. Fragment is a new extendable class available in the Android SDK. Visible interface elements can now be controlled by Fragments instead of Activity, which permits the elaboration of more flexible interfaces. Part of the interface can be changed by replacing one Fragment with another Fragment. Each Fragment is attached to the Activity and maintains access to the Activity. Fragments main intention was to simplify the adaptability of an application between smartphones and tablets where two screens on a smartphone can become a single screen on a tablet due to the size difference. Fragments increase the modularity of the Android applications.

Although there are no defined and named architectural patterns for Android, one can observe that Android SDK already integrates many simple predefined View components such as Button, TextView, ImageView, EditText and also more complex Views such as ListView, GridView, etc. One can also find several Controller-like components such as ViewFlipper, ViewSwitcher, etc. Views can be combined together on the screen using an xml file helping to define a layout and can embed other Views, defining the appearance.

3.5 Mobile architecture related works

The questions of mobile architecture and the mobile development process have been investigated since the first mobile devices, such as mobile phones and Personal Digital Assistants (PDA), appeared.

Several works have been conducted on high-level aspects (software development methodology and project management) aspects studying the appropriateness of Agile methodology[1] and proposing new methodologies for mobile application development: A Hybrid Method Engineering Approach [30], MobileD [31], etc. Among other aspects, the reusability of components was noted as a very important one.

Some authors were focused on web service-based mobile applications. The authors of [32] proposes a Balanced MVC architecture to partition the core of the application optimally between client and server for different types of application. In [33], the authors study the gap in mobile service-oriented application and propose a mediator layer between the mobile device and the server to fill the gap. Those authors do not include any concrete client-side architecture.

Other works were conducted on the low-level (architecture) issues. The authors of [18] conducted an experiment on the impact of design patterns on Agile mobile development and proved that iterative development benefits from defined architectures and patterns. The authors of [34] analyzed the possible cases of application of MVC and PAC architectures in mobile J2ME and Symbian development and concluded that PAC is slightly more suitable due to the modularity of the interface. Authors such as [35] propose some guidelines for designing and developing mobile applications based on a single concrete implementation example.

Concerning Android systems, authors mostly concentrated on security and privacy problems rather than on application architecture. The only work on Android development proposes performing communication between all Android components via interfaces [36]. To our knowledge, no other work has been conducted on architectural design patterns on Android.

3.6 Conclusion

Previous researches and development experience proposed numerous architectural patterns: MVC, AM-MVC, MVP, PAC, HMVC. Although those architectures are well-known and widely used for different programming languages, not many works on suitable architecture for mobile development have been carried out. Only a few works study the impact of architectural pattern and design patterns on mobile development. Although the results showed that patterns are beneficial for mobile development, the modern Android system does not impose, explain or integrate clear architectural design patterns. iOS system developers integrated the MVC-like architecture and hides the complexity from developers with the dedicated integrated development environment (IDE). Android developers should choose suitable architecture instead of concentrating on functionalities, security and usability that may decrease the application quality. The quality of the architecture chosen by developers is highly dependent on their previous experience. The EUTECH

[1] Iterative methodology that split the full projects into small tasks and permits to deliver a part of fully functional and tested software at the end of each iteration.

SSII company also reported that Android development is more costly and less evolutive than iOS applications and one of the reasons is the architectural choice.

In this thesis, we bridge the gap of Android client-side architectural pattern choice. The study analyses the existing architectural patterns and its current adoption by Android developers and then propose a detailed architecture suitable for Android applications named Android Passive MVC.

The summary of architectural patterns presented in this chapter is shown in the Table 3.1. Each column in the table represents the architecture in historical order from left to right. Each row gives comparative attributes: components involved into the pattern; hierarchy supports; responsibility for visualization, event interception and presentation logic; responsibility for the data; involvement of core functionalities and, finally, the communication mechanism between the components. The last column summarizes the Android Passive MVC (or Android MVC) proposed in this thesis.

All architectures have similarities, but differ in the way the components are named, the responsibility of each component and the communication between them. All architectures are based on three components with the exception of AM-MVC, which has four components. PAC and HMVC are hierarchical, supporting hierarchical interfaces as well as Android MVC. Most of the architectures do not explain the place of the core logic regarding the components except PAC that explicitly divides all software into cooperative agents where each agent is responsible for a piece of the core logic. Android MVC includes an explanation of Domain Model and its position regarding other components. We observe two types of communication processes in the proposed architectures: components rather communication via Observer-Observable patterns or via an intermediate component and choose the second for Android MVC.

All models presented in this chapter are general and the Android MVC proposed in this thesis joins the family of the state-of-the art models. To distinguish the proposed model from Android, it could be called Hierarchical Passive MVC.

	Classic MVC	**Passive MVC**	**AM-MVC**	**Supervising controller MVP**	**Passive View MVP**	**PAC**	**HMVC**	**Android MVC**
Components	Model View Controller	Model View Controller	Application Model Model View Controller	Model View Presenter	Model View Presenter	Presentation Abstraction Control	Model View Controller	Model View, Controller
Hierarchy	x	x	x	x	x	Triads by control	Triads by controller	Triads by controller
Visual component	View = piece of screen	View = piece of screen	View = piece of screen	View = full screen	View = full screen	Presentation = piece of screen	View = piece of screen	View = piece of screen
Event interception	Controller	Controller	Controller	View	View	Presentation	Controller	Controller
Presentation logic	Controller	Controller	ApplicationModel	Presenter	Presenter	Presentation	Controller	Controller
Data	Model	Model	Model	Model	Model	Abstraction	Model	Model
Core functionalities	x	x	x	x	x	Abstraction, high level triads	x	Domain Model
Communication	View-Model via Observer-Observable	View-Model via Controller	View-Model via ApplicationModel View-AM via Observer Observable	View-Model via Observer-Observable	View-Model via Presenter	Presentation-Abstraction via Control	View-Model via Observer-Observable	View-Model via Controller

Tableau 3.1: Comparison table between architectures

Chapter 4

Mobile permission systems

Abstract

One of the security mechanisms embedded into many systems is a so-called permissions system or an access model providing a subject with rights over an object. We believe that a well-designed permission system can be a proactive tool that assures transparency and security. This chapter provides descriptions and comparisons of desktop and mobile permission systems. The limits of current mobile permission systems are examined in the light of European legal requirements. The detailed state-of-the-art related to mobile permission system analysis, modeling and improvement is presented.

4.1 Introduction

A permissions system is a type of security system restricting or giving a right to access a piece of data or a service and is generally represented as an Access Model where a Right over an Object is assigned to a Subject.

Traditional desktop systems embed file permission security where a right to read, write, delete and execute a file or a folder can be given to a particular user (file owner), group of users or to the public. All programs have the same right as the logged-in user. Mobile systems modify the logic of traditional permission systems considering that a mobile device can only be owned by a single user. Therefore, rights are assigned to individual applications instead of users. Mobile phones also possess particular capabilities such as phone calls, camera, sensors, etc., and, therefore not only sensitive files are protected with permissions, but also the application programming interfaces (API) exposing sensitive services. The summary of desktop and mobile permission systems represented as an Access Model is given in Table 4.1.

Mobile systems often build a mobile permissions system on top of a traditional file permission system that allows sandbox security: each application is considered as a single user that only allows one private folder to be accessed; assigning groups of applications allows data to be shared between applications; additional mobile permissions allow an application access to other data or services.

	Desktop	Mobile
Right	Read, Write, Execute, Delete	Access, Read, Write
Object	Folders and files	Sensitive data: folders and databases tables, Sensitive API
Subject	User, user groups, public	Application

Tableau 4.1: Desktop and mobile permission systems comparison

To mention some early mobile systems, Symbian OS embedded permissions for sensitive services that were assigned to each application if needed. At earlier stages, Symbian notified users about requested permissions and assumed the users would be personally responsible for installing only reliable applications with legitimate permissions. As users were not able to judge applications, Symbian, subsequently included professional verification and mandatory signatures for sensitive rights requests. The Symbian system was able to run non-signed applications that did not need any particular permission, signed or system applications with limited capabilities and system applications having full access [37].

Java Platform, Micro Edition (Java Me or formerly J2ME) is a Java platform designed for mobile phones, personal digital assistants and other embedded systems. The Mobile Information Device Profile configuration for Java ME is dedicated to mobile device usage and integrates a permission-based security mechanism. By default, only network connection-related permissions are available, such as permission to connect to a socket, open http or https connections or establish ssl connections, but other permissions could be integrated. All applications are run in one of four security domains created by manufacturers: untrusted, trusted, minimal or custom.

- Untrusted domains prompt the user when a permission is required
- Trusted domains grant all permissions automatically when required
- Minimal domains reject all permissions automatically
- Custom domains can specify if the permission interaction mode is 'allowed' - permission should be granted automatically or 'user' - permission should be prompted. An example of prompted permission is shown in Figure 4.1.1.

All permissions that are not mentioned in the domain configurations are rejected. Permissions have three granting modes: oneshot, session and blanket.

- Oneshot means the user's decision will be used only once and will not be remembered
- Session means the decision will be remembered until an application is closed
- Blanket means the decision will be kept in the memory until the application is de-installed

Mobile applications named MIDlets are installed into different domains based on their credentials: third party identified, non-identified, manufacturer, etc., thereby protecting users from malicious third party applications.

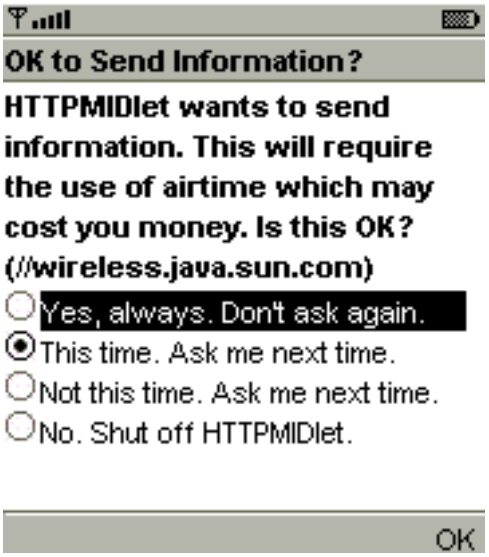

Figure 4.1.1: Java ME MIDlet asks for a permission.

Modern mobile systems such as Android and iOS also include mobile permission systems. The iOS default systems can run only applications verified and signed by Apple. The iOS platform allows non-native applications access to the functionalities listed in privacy settings: location services, contacts, calendar, reminder, photos, microphone and Bluetooth (sensitive data, such as SMS and e-mails are not shared at all) and sends push notifications. The connection to Facebook, Twitter, Flickr and Vimeo was added to the platform on iOS7. Apple controls and prohibits the abusive use of those APIs, accepting in the Apple Store only the applications that passed the quality control. Some permissions, such as contacts and calendar access, are automatically granted by Apple verification, but other permissions such as location, push notification and Facebook access, require

he user's decision. Developers should request permission in the code before the actual permission usage; the application displays a pop-up explaining what permission is needed when the code is reached. Developers can also add a custom text to each permission to clarify the permission's usage adding the custom message into Info.plist using corresponding permission keys, but this is not mandatory. An example of permission request on iOS is presented on the left in the Figure 4.1.2. The user can accept or decline the permission. If permission is declined, the corresponding action is not executed. If the permission is accepted, the application obtains access to the corresponding API. The user is asked to grant permission only once, but he can enable or disable such permission for each application in privacy settings integrated by default into the iOS.

Android applications are not certified by Google and do not need any particular signature to run on the Android system. To cope with malware, Google introduced the 'Bouncer' security feature that can verify any application just before its installation on the device. Bouncer also scans Google Play for possible malware, but can be bypassed. Concerning non-explicit malware applications, users must check all available information themselves to judge if an application is legitimate and not potentially harmful or abusive. The user is fully responsible for installed applications and Google only provides useful information to help make the judgement: description, comments, notes and permissions required.

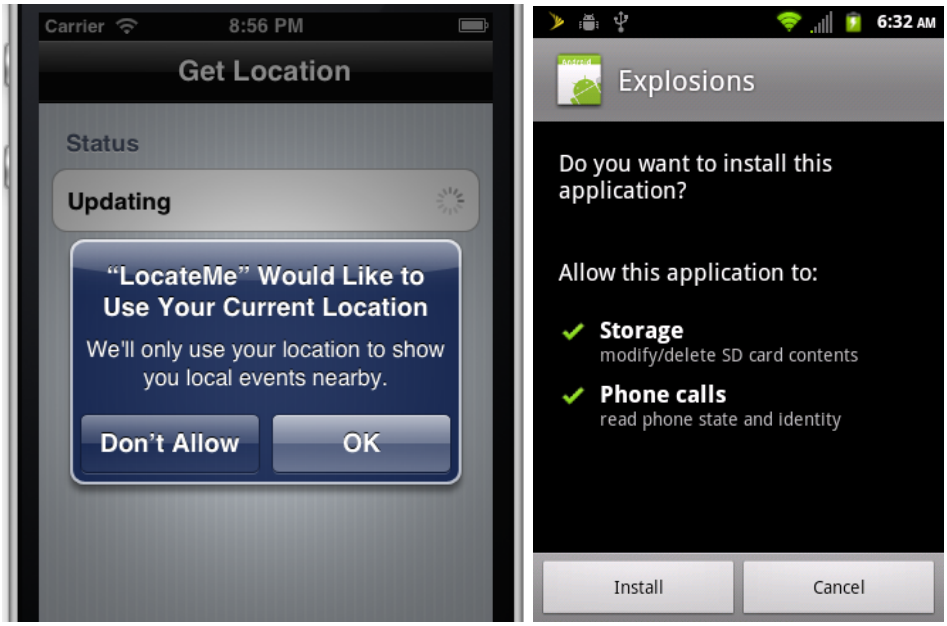

Figure 4.1.2: iOS (left) and Android (right) systems ask for permissions.

The Android system remains a sharing principle: applications can have access to all native applications' data and services and can share the data themselves. Android has a predefined list of permissions that developers can use. According to the extract from Nexus 4 in November 2014, Android 4.4 contains 229 permissions of different levels: 30 normal, 48 dangerous, 11 development, 70 signature and 70 signature or system permission. Third party application may only use 89 Android permissions as signature or system permission are only intended to be used by manufacturers. If an application needs access to a protected interface or data, developers should explicitly add permission into a dedicated file called Android Manifest; this way, each application is associated with a list of permissions. Normal permissions will be automatically granted during installation,

the user will be prompted about dangerous permissions, development permissions will be granted if the corresponding developer option is active on the device. An example of permission prompt is shown on the right in Figure 4.1.2. The user has to accept a full list of permissions before installing an application and no permission can be revoked during or after installation.

Android did not initially include a iOS-like default system permission manager (privacy settings), therefore the user had to enable or disable the entire functionality to disable access to related data (e.g., Wi-Fi or 3G for Internet connection; GPS for geolocation) or to use additional privacy enhancing applications. This feature only appeared with Android 6.0 Marshmallow in September 2015.

Native Android permissions have similar prefixes: 'android.permissions.{*}'. For example the permission for the Internet access is defined as 'Android.permission. INTERNET'. Only five Android permissions are prefixed by 'com.android.{*}': browser, alarm, launcher and voicemail related permissions. Custom permissions that developers define to protect new interfaces to share services or data can be named freely: Google does not impose any naming rules.

Table 4.2 sums up the iOS and Android permission systems. For each system, we present the goal of the permission, the number, the type of application verification, notification, granting and revocation mechanisms as well as the types of rights protected by permissions. The last line represents the requirements of privacy policy.

	iOS	Android
Goal	Notification and control	Security judgement
Number	10	89
Verification	Apple certification	Verification for malware
Notification	Some permissions by pop-up	Dangerous permissions on pre-installation
Granting	Apple or in-context	All-or-nothing, pre-installation
Revocation	Individual, in-context, settings	All-or-nothing, de-installation
Rights	Access	Access, Read, Write
Privacy policy	Required for children's apps	Not required

Tableau 4.2: Comparison of iOS and Android permission systems.

4.2 Limits of modern mobile permission systems

Permission systems are embedded into mobile modern operating systems and manage rights for all installed applications. In theory, the permission system should help applications to comply with the law and provide Privacy by Design. In this section the permissions systems of Android and iOS are evaluated regarding Privacy by Design principles and European Directive regulations.

- It states, that users should clearly understand what data is used, how and for what purpose. The purpose should be clear, explicit and legitimate.

The authors of [38] analyzed 1 application from 10 categories and concluded that Android's permission system lack user's consent and does not adequately protect user's private data: users need a simpler solution to make decisions about privacy and the security of Android applications.

It is a recognised fact that users do not understand many default Android permissions and fail to judge the application privacy and security correctly using the permission list [39, 40, 41]. In [39], the authors found users often do not understand the terms used in Android permissions (e.g. phone state, phone id, coarse location, account authenticator, etc.) and are unable to judge the security and privacy aspects of displayed permissions. Even people who are knowledgable about information technology and experienced at using Android had difficulties understanding many permissions. Many users do not understand why displayed permissions are needed for the application to function and cannot link the functionality with a permission. The authors found users do not have enough background to judge applications, moreover many users reported they thought all the applications on the market were monitored and trusted.

In [41], authors studied the usage and comprehension of ACCESS_WIFI_STATE Android permissions and found the the users do not understand the privacy risk related to this permission. The same study shows, that ACCESS_WIFI_STATE is used by some application to geolocalise users without requiring the dedicated permission.

In [40], the authors performed a survey and a laboratory study to see if users pay attention and understand Android permissions. The attention and understanding rates were low for both studies, but a very small group of participants showed good comprehension of permissions. Only 3% of participants could correctly identify 3 shown permissions. A high proportion of the laboratory study participants did not know about permissions at all. This study confirms that most users cannot judge Android applications with the help of the current permission system and do not link a particular permission to a possible risk: the current permission system lacks this information.

The authors of [42] studied the effectiveness of on-installation permission based systems on Android. In principle, the list of permissions should warn users against installing malicious applications, the impact of application's vulnerability should be limited by permissions and the application review ought to be simple and concentrate on dangerous permissions. The authors found that the majority of Android applications have at least one dangerous permission. Many of them allow access to personal information and to the Internet at the same time, which can potentially generate data leakage. Manual verification showed excessive permissions added in error and the functionalities that could be developed with less permission. They also point out that during installation the user is prompted about dangerous permissions and will not pay attention to frequently asked permissions such as Internet.

Those studies show that the Android permission system is failing to provide adequate information about data usage. Moreover, both iOS and Android default permission systems usually inform us about data access, but not about any other data processing. For example, no permission is needed to transfer the data. Android and iOS include permissions for functionalities that can be related to personal data transfer, such as Bluetooth and Internet, but there is no indication of whether personal data is involved in a transaction; Internet access can be either harmless or harmful depending on the context.

Android and iOS permissions do not include purpose explanations. An iOS application

helps users to understand the purpose by asking some permissions in-context, but if an application has been granted permission once for one functionality it could use it again for a different purpose without informing the user. Moreover, the developer is not obliged to ask for a permission just before the action and in the right context, the only condition of the platform is to request the permission before the actual permission use. iOS developers have the possibility of adding a custom message to the permission requested, but it is not compulsory. Developers also tend to add unclear explanations about the purpose such as "To improve the service" or "To improve the user experience". Android users can only guess what the permission is used for and whether the use is legitimate.

- Users should explicitly agree with a permission and to have a possibility to disagree and revoke permissions.

Privacy for mobile users does not necessarily mean anonymity, but rather the possibility of a choice of what to use, keep and share and when to stay anonymous [43, 44] and to have a control over their data.

The control over permissions is very limited on Android. It has been shown that users tend to install Android applications regardless of permissions, as they want the particular service; users know that all permissions must be accepted to obtain it [45]. The Android permission list looks like a license agreement on a desktop application which everybody accepts but very few actually read [46].

Some iOS applications request many permissions on the first launch of applications, therefore many pop-ups appear one after another. The repetition of notifications leads to the fact the users do not read them. Moreover, until recently, Apple authorized the use of device identification: this identification number was not considered private. Many applications used this number to uniquely identify their users, therefore, many applications were considered privacy intrusive [47].

- Data treatment should be limited and adequate regarding the purpose. The data storage should be limited and the minimization principles should be applied.

The authors of [48] studied the user perception of an application having permission to access data at different moments of application usage. They found that if an application needs user action to perform something (e.g. send sms), users expect the application to only demand/request permission to perform an action when the user-event occurs. There is one exception: users are less sensitive to geolocation access and assume applications have permanent access. Currently, Android and iOS do not add any conditions to granting permissions.

Neither permission system is very fine-grained. First, there are neither conditions, nor limits assigned to permission usage once the permission is granted. Second, one permission, such as address book access permission, actually manages a lot of data such as full names, phone numbers, addresses, etc, where the application probably only needs the full name to function.

4.3 Mobile permission analysis

This section gives an overview of studies related to Android and iOS permission analysis. Subsection 4.3.1 presents studies related to permission list analyses of Android benign and malware applications. Subsection 4.3.2 presents Android permission use analysis

tools. Subsection 4.4 details works dedicated to helping users to verify or to configure Android and iOS applications. Subsections 4.5 detail all improvement to current permission systems already proposed by the state-of-the-art. Subsection 4.6 presents static and dynamic analyses tools permitting analysis of the data flow in Android applications. The final section presents some other permission-related works that did not belong to previous subsections.

4.3.1 Permission request analysis

Many works analyzed permissions required by applications. The analysis of malware applications aimed to define permission patterns that help malware detection. The analysis of benign applications aimed to find correlations between permission requests and other parameters such as is price, category, rating, etc., and also to compare malicious and benign application requests.

4.3.1.1 Benign applications

Several studies have been made on Android permission analyses.

In [49], the authors analysed the Android permissions of the top 50 free Android applications of 2009 using Self-Organizing Map (SOM). Results show that applications of same category do not necessary require same permissions due to different functionalities but similar sets of permissions were identified for applications from different categories. Some permissions are very highly used, and some are rarely used. The most used permission is for Internet access. The authors also identify sets of permissions used together, for example “fine location” and “coarse location”. They found custom developer’s permissions are rarely used, they propose working on another access restricting system for application components, rather than including custom permissions in the permissions system. They propose identifying a vocabulary for defining permissions systems to simplify developers’ work and enable finer grained permission definitions.

In [50], the authors analyzed permissions the most popular and newest Android applications from official and non official markets, as well as Facebook applications and Google Chrome extensions. They found that popular applications request slightly more permissions (possibly due to making more functionalities available) than new applications and applications proposing a privacy policy request a slightly lower number of permissions. Applications that do not have any associated developer web-site are observed to be more intrusive. Free applications on average request more permissions than paying apps even if permissions regularly used for advertisements are excluded from the study. Applications that have look-alike names to popular applications request more permissions than the average.

Several authors worked on permission usage patterns identification on Android. The authors of [51] used probabilistic methods to identify patterns for high and low ranked applications. They use unsupervised learning to identify 30 Android permissions patterns. The patterns were more or less represented in different categories. The authors also note that applications with a low rank in the store often deviate from identified patterns.

In [52], the authors used 999 Android applications to build a graph based on co-occurrence of permissions in different applications by category. The most frequent groups for each category were identified by a modularity optimization classification algorithm, and considered as legitimate requests for an application in a given category. The authors compared those groups with dangerous permission combinations from [53] and found the

combinations in some groups presumed to be legitimate. The authors note that the cluster definition was unclear and the methodology would benefit from the integration of different weightings for different permissions. A potential bias is based on the fact that the most popular permissions were found in most groups.

4.3.1.2 Malicious applications

In [54], the authors propose scoring Android applications using permissions and Naive Bayes Model, Naive Bayes with Informative Priors and, finally, a Mixture of Naive Bayes with categories using Latent Dirichlet Allocation (LDA). The results in malware detection are improved compared with [55] and [53].

In [56], the authors extracted permissions and API calls from benign and malicious applications and ran machine learning techniques for malware detection: SVM, RaJ48 Decision Tree and the Bagging algorithm where the Bagging shows the best results.

In [57], the authors analysed the use of the permissions in a set of 1,227 clean and 49 families of malicious Android applications. As well as manifest permissions, the authors used the available *Andrubis* tool [58] to recover used permissions for their analysis: some permissions may be declared but not used by an application. The authors statistically analysed and compared malicious and clean applications and the corresponding used and required permissions. The results showed that the most frequently used Android permissions are quite common to both application sets, but some permissions are used uniquely by malicious apps, some only by clean apps. Finally, the possible patterns of 2 to 4 permissions were automatically generated and an occurrence frequency was assigned to each pattern. The 4 permissions limit is assigned to the patterns due to the calculation costs. Some unique patterns for malware detection were identified. The authors did not extract permission patterns from clean applications.

Recent work [59] has proposed using statistical methods such as the correlation coefficient, mutual information and the t-test to identify the top 40 risky permissions using datasets with benign and malicious applications. The authors performed clustering techniques to identify patterns and detect malicious applications.

4.3.2 Permission use analysis tools

Some authors analyzed how required permissions are used by applications. In [48], the authors analysed permissions of the top 100 Android applications and found that most permissions are used only in response to the action made on GUI. Many permissions are used only for display purposes. Only 5% of applications legitimately require permanently granted permissions.

The authors of [60] propose a permission use analysis tool called *Permlyzer* that not only verifies what permission is actually used by an application, but also detects what Android component (Activity or Service) is responsible for permission invocation as well as what class, method and user interface components and events are related. *Permlyzer* also reports what API calls result in a permission check which can reveal that a fine-grained functionality such as getLastKnownLocation() call is more precise than the corresponding ACCESS_FINE_LOCATION permission. Furthermore, the tool detects commonly used permissions or recurrent permission use that can uncover some functionalities. They also analyzed Android permissions use in normal and malicious applications with *Permlyzer*. Results show that malicious applications tend to call protected API in the main Activity and also during the components creation. Knowing that Android applications can be

launched automatically, it shows that malicious applications can execute malicious behavior without any user interaction or awareness. They also found 537 correlations between permissions and also discovered the correlations between the same permission used many times. The authors found that the device ID (IMEI) and location are generally recovered without user interaction in free and not explicitly malicious applications. Finally, they found that 4.2% of free applications in the dataset have similar behavior to malicious applications.

Some works shows the abusive permissions declarations by Android developers. The authors of [61] built a *Stowaway* tool analyzing the source code and detecting the permissions it requires. They found that developers often declare permissions they do not need: breaking the principle of least privilege. The main reason for permission overuse indicated by the authors is poor documentation: many permissions are not mentioned, some permissions are required by the documentation but do not actually exist, some information about the needed permissions is confusing. Developers often use deprecated permissions for retro compatibility or due to outdated sources.

The authors of [62] use the *Stowaway* tool built by [61] and checks the correlations between the misused permissions, API calls related to those permissions and questions on StackOverflow[1]. They found that many permissions are still misused (even more than reported by the previous study [61]) and the misuse does not depend on the related API calls. They also noted that the more popular a permission is (often used by applications), the more questions related to this permission are found on StackOverflow and the less is the misuse. They suppose that StackOverflow bridges the gap of documentation over time proposing various examples.

4.4 Permission-based decision support systems

The authors of [55] proposed a risk warning system based on the occurrence of a total of 24 permissions (manually identified by authors as dangerous) in each category of applications. A warning is issued if an application requires one of these permissions or two dangerous permissions that are used by less than a certain percentage of applications. The authors applied their risk metrics to applications collected from Android Market and a malware dataset, and found that malware triggers many more risk warnings than benign applications. The results for category-based risk warnings were similar to overall occurrence-based warnings.

The authors of [63] created a crowd-sourced system named *ProtectMyPrivacy* that collects and analyses iOS user's configurations for different applications and returns privacy recommendations about accesses that are necessary or unnecessary for the functionality. The system is based on crowdsourcing and proved effective event with only 1% of users being experts in the domain. The system was tested by 90,621 real users of jailbroken phones on over 225,685 different iOS apps. Smartphone's unique device identifier (UDID), address book, location and music can be protected by the system returning shaded data. The system is made extendable for further private data support.

Interestingly, many developers complained they did not access the address book as it was detected by *ProtectMyPrivacy*. Investigation showed that the advertisement library was downloading address book information and developers were not aware of this. A light

[1] One of the most popular crowdsourcing platform where developers ask and answer development-related questions

version of the system receiving recommendations about known applications is available for non jailbroken devices.

The authors of [64] proposed searching for a justification of permissions usage in the application description with natural language processing techniques and warning users if it is not found. The proof of concept was done on three Android permissions. Further work improved the detection and number of supported permissions [65].

Recent work proposed a recommendation system balancing the rating of applications with privacy issues such as required permissions [66]. The authors build an application risk score based on the number of required permissions and the occurrence of those permissions in the category: the more permissions used in the category, the lower the penalty incurred by this permission; the more permissions required by an application, the greater the risk of that application.

4.5 Improvement of modern mobile permission systems

Many studies aimed to improve the current permission system by adding the possibility of revoking permissions or granting them conditionally.

4.5.1 Revoke permissions and mock data tools

Many works concentrated on improving Android with an all-or-nothing approach by revoking Android permissions.

MockDroid is a modified Android OS allowing to mock the data (fake, empty or unavailable) requested by the application giving user the possibility of revoking Android permissions [67].

Flex-P also permits users to revoke Android permissions. The system adds check boxes to permission groups shown to users before installation, and also displays the same list on the screen with the information about installed applications [68].

TISSA proposes the possibility of controlling the access to phone identity, location, address book and call logs. Users can choose if an application can access the data or should obtain the anonymized, empty version of fake data [69].

ProtectMyPrivacy can return shaded data instead of real one on iOS [63].

All these propositions modify the platform to integrate new functionalities. This functionality was integrated into Android 6.0, introduced in September 2015.

4.5.2 Conditional granting

The authors of [70] worked on improving the J2ME permission system to make it possible for the user to add additional constraints to accepted permissions, such as a number of sms or mms that can be sent by time unit (day, week, month) and a limit on downloading using Internet. They proposed the extended J2ME (xJ2ME) that can process new fine-grained policy written on Security Policy Language (SPL).

Apex permits Android permissions to be revoked and adds conditions for permission usage, such as number of usage per day [71]. Users set permission constraints on applications at installation time.

Saint adds policies to current Android custom permissions to restrict permissions usage to applications that already have particular permission or have a trusted signature (white lists) [72]. The authors provide a tool repackaging android applications to work with

the new permissions and also provide a tool verifying the use of new permissions. All permissions are embedded into the verification tool. Users could also add new permissions via provided GUI.

In [73], the authors propose an ontology where an actor is given the right to take an action on data. Rules are defined using SWRL language. Complex rules forbidding or allowing the data access on a particular condition can be added to the policy. It is not clear what other action than "access" is supported by the ontology. A firewall was implemented on Java and ported to Android. The perception of users and the applicability to real applications is not discussed.

The authors of [74] propose a *CRePE* system permitting a list of access rules (grant or revoke permissions) to be enforced for a resource using the context of phone usage: time and location. Policies can be defined by users via a user interface or a third parties through sms, wi-fi or bluetooth. The system is implemented as a middleware between the Android permission verifier and an application.

4.5.3 Usage model and Digital rights management (DRM) technologies

DRM technologies restricts the file usage and can be similar to permission systems despite the different goals. DRM systems restrict the file access on certain conditions and mainly protects intellectual property and commercial file usage and distribution.

The $UCON_{ABC}$ is a usage control model combining the Access Model but also adding a DRM conditional to control not only file access rights but also their usage [75]. This model contains a decision module granting or revoking a Right over an Object to a Subject based on attributes attached to both the Subject and the Object (such as price to open a file and an amount of money available on the user account); obligations that are mandatory pre-requirements (such as license agreement acceptance) and conditions (such as current time regarding time period for accessing the file). Attributes can be mutable or non-mutable: fixed or modifiable by the system. This model is very general and can express either permissions or DRM, although it is more DRM-oriented.

The authors of [76] propose a $UCON_{ABC}$ model based system named *T-UCON* to enforce file policies across applications such as pay-per-use and N-time access policies. *T-UCON* is made as a Java Virtual Machine extension to intercept file access calls and verify policy compliance.

4.5.4 Finer-grained mobile permissions

Few works propose integrating finer-grained permissions. In [77], the authors propose replacing some Android permissions with finer-grained permissions. They replace the permission 'Internet' with a new permission specifying the domain to which the application is to connect, the new permission is named InternerURL(example.com). They introduce 2 permissions for working with AdMob advertising service: AdsPrivate - sending only a developer id to the server for advertisements - and AdsGeo - sending the user's geolocation alone with developer id. Permissions for geolocation are replaced with a new permission whose method only gives an approximate to within 150 meters. They replace 'write settings' permission with 'setRingtone' permission, and 'read contacts' with 'read visible contacts' permission. Finally, permission to access the unique device identifier is replaced with a method returning a randomly generated unique identifier.

The authors of [78] propose integrating additional *'Flow permissions'* corresponding to the data transfer permissions. They propose linking data related permissions such as access to sms, calendar, contacts, etc., to 'sink' permissions such as Internet, logs, sms, e-mail and sd-card.

Several works try to redefine permission systems. The authors of [48] (University of Washington and Microsoft Research) propose rethinking the current permission granting mechanism using permission lists (Android) or prompts to users (iOS). They introduce user-driven access control where a permission is automatically given as a respond to a user-event. 'Access Control Gadgets' are introduced, these are predefined user-interface elements permitting certain permissions to be granted (for example: take photo button with access to a camera). The granted permission can be 'one-off', limited-time (session), scheduled or permanently. That allows in-context punctual permission granting, assures least-privileges and reduces attack vectors. They envisage the possibility of linking hot keys or a vocal control to permissions in the same way as *'Access Control Gadgets'*.

The authors of [79] propose a per-data permission system: more fine grained than android default permissions. Access permissions are granted to data stored in SQLite databases and the access can be restricted to a piece of data or a column (only phone number, only names from contacts) or to groups or rows (only work contacts can be accessed) and mixed or a cell. Privacy policy is expressed with subject having (1) or not having (0) an assess to an object. The prototype was implemented on Android and included only contact list data access management.

PSiOS is a privacy granting policy enforcement tool for the iOS system where the user or enterprise can define custom and fine-grained policies to control the use of all Objective-C methods and API calls [80]. Per application policies are described as an xml file, where the name of the method and arguments are added with the policy type: log the usage, deny the usage, allow the usage but return shadow data. The exact name of the API or method has to be known by a policy administrator to make the policy. A tool is implemented as a shared library and can be used on jailbroken devices only.

4.6 Code analysis and data flow control

Many studies propose tools to perform static and dynamic analysis of the Android applications' code. The main idea is to control the data flow and identify or prevent data leakage.

4.6.1 Static analysis

To cope with unnecessary permissions, the authors of [81] propose a permission check tool informing a developer about unnecessary permissions. The tool inspects the source code, searching for known methods requiring permissions and compares the list obtained with permissions already defined by the developer. The developer is then notified about missing or excessive permissions.

Kirin tool analyzes manifest files to identify dangerous permission combinations to flag potential malware before its installation [53]. Two dangerous permissions and some combinations of permissions are identified by authors and added to the installation privacy policy used by *Kirin*.

SCanDroid (University of Maryland) [82] is a static analysis tool permitting the certification of the application's security. *SCanDroid* performs data flow analysis of Android

applications linking permissions use and flags some data flow as potentially dangerous.

The authors of [83] performed a static analysis of iOS applications to monitor the private data flow on iPhones. They detect access to private data through the defined methods calls and trace the use of the data. The authors found that applications on AppStore and even the applications from the alternative market Cydia operated with respect to the user's privacy and very rarely transmitted sensitive information without user awareness. As an exception, the UDID was often used and transmitted (especially to advertisers) due to the fact that Apple did not consider this information as private. In 2013 Apple forbade the usage of the unique device id by developers.

Similarly *AndroidLeaks* [84] (University of California) detects the possible privacy leaks on Android by automated static analyses. Android permissions and corresponding API calls are analyzed for possible sources of private data and sinks (outgoing transmissions). Private data is tainted and followed till the sink. Applications need to be decoded from Android Dalvic to Jar and then a call graph has to be built. The authors found Android application leaks a lot of data, especially apps containing advertisement libraries leaking phone id, location, SIM number, etc. Similar results were obtained by the Mobilitics project [85].

In [86], the authors proposed a *ScanDal* static analysis tool to detect privacy leakage within an Android application. They identified many privacy leaks in existing applications.

Further works *ComDroid* [87], *Epicc* [88] are static analysis tool focused on the detection of inter-component communication vulnerabilities. *DroidChecker* concentrates on privilege escalation attack detection [89].

The authors of [90] propose a static analysis-based framework for privacy data leakage detection. The framework is based on a security type system and a defined privacy policy where certain methods can only use data of a defined security type.

The authors of [78, 91] propose a static analysis tool named *Blue Seal* to monitor the data flow between sources (Network, e-mail, IME, SMS, mic, calendar, accounts, sd-card, contacts, camera, call log and sim-card) and sinks (network, email, sms, sd-card and log) automatically defining flow permissions. *Blue Seal* also detects inter-applications data flows.

Amandroid [92] is a static analysis framework detecting data leakage through inter-component communication and vulnerabilities such as non protected authentication and intent injection.

FlowDroid [93] is a static taint analysis tool with high data leakage detection.

Those automatic solutions can help with the manual investigation of an application and its certification.

4.6.2 Dynamic analysis

Some works propose systems to improve the visibility of private data use for users. *TaintDroid* is an Android dynamic analysis data flow control system [94] that taints the data in Dalvik VM and follows it to the sink (e.g. Internet). *TaintDroid* found many application leaks of personal data without users being informed (The End User License Agreement was often missing or uncompleted). Half of tested applications sent information to third-parties' publicity servers even if the advertisements are not displayed. The authors of [95] analyzed the RATP Android application using *TaintDroid* and also dynamically analysed its iOS version. The authors found that an application collects a number of private data

and personal identifiers even if the privacy policy of this application claim to not collect any of it. Application also bypass the Apple restriction for advertising using user's personal identification instead of the dedicated id that can be controlled by the user.

AppFence is an improvement on the previously developed *TaintDroid* [96]. *AppFence* proposes monitoring the private data flow and preventing transmission of data tagged as 'device only'. In addition, *AppFence* can return shaded data (empty file, empty lists, generated data, fixed numbers) to applications instead of real private data (device id, location, camera, logs, sms, contacts, etc.).

The authors of [97] propose a solution for monitoring the private data flow on Android without any modification to the Android system. The monitoring module is injected into each application. The module search for calls of particular sensitive methods (get contacts, send sms, etc.) and report the usage. The code injection fails regarding some applications.

4.7 Other works

This section presents some other works related to the Android permission system.

4.7.1 Model

In [98], author models Android permissions using a state-machine. Author defines an application needing an authorization (permission) and an application components inheriting application permission. One component at time can be on foreground. One state is defined by the existence of foreground component, by the list of installed applications and by the list of permissions granted to each application. The state is changed by the lunched or terminated component or by an installed or uninstalled application. Some security properties are also defined: such as "installed", permission "requested", permission "granted", permission "authorized" for the application. Using those properties the authors define security conditions for the application.

4.7.2 User interface

Some authors work on user interfaces and visualization of permissions. [99] have tested the effectiveness of different permission visualization methods, including Android, Facebook, Health-Vault and Twitter. They observed some systems only choose to include the access to the resource permission (classified as a "resource"), others also include the action that can be executed on the resource, such as display, modify, change, etc (classified as "resource-action"). All permissions can be grouped by action or by resource. The display can be organized in the outline, paragraph or table with images and icon. Authors found people more easily absorb icons and images than simple text in permission displays. Permissions grouped by action were absorbed more rapidly. Participants were asked to note the best design for absorbing and searching the data and the highest scores were given to icons and images. Users wanted icons, images and tables to be available in programs. The paragraphs (Facebook) and outlines (Android) were disliked by participants.

4.7.3 Permissions as an attack vector

Some authors show vulnerabilities that can be exploited in Android permission systems. The authors of [100] show the permission re-delegation attacks when one application takes advantage of permissions and exposed interfaces of another application. In [101], the authors show the type of attack attack where two applications can define permissions with the same name but with different security levels, therefore one application requiring permissions with lower security levels can bypass the system permission as a permission of the same name has already been granted.

4.8 Conclusion

Access Control Model or permission system is a known security mechanism that was not only already embedded into desktop systems, but also into older mobile systems such as Symbian and J2ME. Although, many systems embed permission systems, the implementation differs from system to system. Modern systems did not take the experience from previous systems and works, and, as a result, iOS and specially Android permission systems have many weaknesses and are highly criticized. Permissions themselves are coarse-grained and not clear, their purposes are not always comprehensible by users, permission control is limited as well as data protection. Numbers of works show the confusion of users regarding modern permission systems.

Much research has been done on mobile permissions. Permissions usage of both benign and malicious applications were analysed to discover correlations and malware-unique patterns. Although no work has been conducted to define permission patterns of benign applications to detect malware. Some works proposed scoring mobile application risk using permissions or provided permission verification mechanisms. A large number of studies were focused on code analyses: static and dynamic. Such works showed many Android applications leak, or may leak, personal data without user awareness (lack of security) and propose monitoring all the data flow, although the adoption of such technologies by users was not tested by any of the studies.

Some works proposed improving current permission systems by revoking permissions (returning fake or empty data) or by adding custom or context-based conditions on permission granting. Few works proposed integrating new permissions into existing mechanisms, such as flow permissions (transferring the data), Internet access permissions restricted to a particular domain; advertisement permissions or a per cell/column/line permission for the data stored in SQLite database. Finally, few works propose re-designing permission systems and modern mechanisms. The PSiOS tool restricts the usage of any method call instead of integrating a limited number of permissions. Another author proposed rethinking the permission granting mechanism by introducing trusted pieces of the interface that would grant a one-off permission in response to the user-event.

In spite of the the amount of investment and research done on mobile permission systems, the research topic remains open in terms of permission definition, granting mechanism as well as permission visualization and technology adoption by users.

Part II

Quality and Security

Chapter 5

Development of mobile applications of quality: Android Passive MVC architectural pattern

Abstract

Nowadays, the demand for mobile application development is high. To be competitive, a mobile application should be cost-effective and should be of good quality. The architectural design pattern choice is important to ensure the quality of the application on the design, development, integration and future update phases and to reduce development time. Two main leaders are very represented on the mobile market: Apple (iOS) and Google (Android). The iOS development is based on the Model-View-Controller design pattern and is well structured. The Android system does not require any model: the architecture choice and the application quality highly depends on the developer experience. Badly written code can not only be buggy and hard to maintain, but can also affect security. Heterogeneous solutions slow down the developer, while the one known design pattern could not only boost development time, but improve the maintainability, extensibility and performance of the application. This work investigates widely used architectural design patterns and proposes a unified architecture model adapted to Android development. We provide implementation examples and test the efficiency of the proposed architecture by implementing it on real applications.

5.1 Introduction

The mobile market has grown rapidly in recent years. Many enterprises propose their services with mobile applications, and the number of mobile application available to users increases every day. Compared to computer programs, mobile applications often have limited functionalities, shorter shelf life and lower price. New applications should be developed fast to be cost-effective and updated often to keep users interested. The quality of the application should not be neglected, as mobile users are very pernickety and competition is stiff.

The demand for smartphone application development is high especially for the two market leaders: Google (Android) and Apple (iOS). While developing application for both platforms, developer can choose either to develop two applications using dedicated technologies or opt for a cross-platform tool that would compile the same code for different target devices [102].

Cross-platform solutions, such as PhoneGap [103], Rhodes Rhomobile [104] and Titanium Appcelerator [105], are often web-based and are embedded into mobile applications as web-pages. Such solutions can have performance issues, use interface that is not platform-adapted and limited functionalities. Native solutions enable use of all the platform's options with better performance and lighter code enabling the creation of an application adapted to the platform.

This work is focused on native Android development and do not deal with cross-platform propositions. First, mobile applications are not meant to be developed using web technologies that can be considered as a hook to get around the dedicated technologies. Second, cross-platform are third party solutions that could be abandoned, for example, if the native solutions forbids the usage of embedded web-pages.

Mobile applications development is totally different between Android and iOS. Android is Java-based while iOS uses Objective-C and, recently, Swift. The iOS integrated development environment (IDE) guides developers to follow an architecture predefined by Apple. Developer can hardly deviate from the guideline and many of the architectural aspects are hidden from the developer by the IDE. Therefore, iOS developers uses the imposed Model-View-Controller (MVC) design pattern even if they do not realize it [106].

Android requires no particular architecture [107] – developers should choose and follow a suitable architecture for their applications. This adds a level of complexity to the Android development process as the developer should not only be focused on the final product, but also on the architecture and code organization. This task can be especially difficult for less experienced developers. Complex applications that do not follow any architecture can end as a 'big ball of mud': incomprehensible and unmaintainable [108]. Moreover, complex code could not only lead to bugs due to testing difficulties, but can affects the security of an application [109].

In spite of the fact that the Java programming language used for Android application development is well-known, the code organization and architecture choice are not trivial for Android developers. EUTECH SSII - web and mobile application development company - reported that the development and evolution of equivalent applications is more time-consuming and costly for Android that for iOS. They noted, that several Android applications needed multiple hard refactoring work to integrate updates and maintain performance while the maintenance of similar iOS application was done smoothly.

Android development books and tutorials are mostly focused on Android SDK technical details and user interface design. Only a few works have been dedicated to the Android

application architecture, while the Android community identifies an architecture as an important part of successful system design and development. Developers open many discussions about suitable Android architecture on forums, blogs and groups.

This study aim to define an architecture and a set of design patterns especially suitable and adapted for Android application development. This chapter presents the research methodology and the proposed architecture named «Android Passive MVC»: an hierarchical architecture based on Passive MVC. The proposed architecture join the family of HMVC and PAC but have different communication mechanisms. This chapter gives the general overview of the proposed architecture, explains and names the components involved and the communication mechanisms between components. The chapter includes an example illustrating the architecture implementation using simple Java classes. An important part of the chapter includes the implementation examples using Android components (Activities and Fragments) and a set of design patterns for some typical situations can be met during application development. Finally, the possible extensions of the basic architecture are presented. Although, the explication and examples are mostly focused on Android system, the architecture is general similarly to the state-of-the-art architectures and could be applied to other systems and programming languages including the currently unknown.

5.2 Research methodology

This section detail the research methodology that led to the proposition of «Android Passive MVC» and related patterns.

The first step was the literature analysis. First, we analyzed the existing architectural design patterns to understand well their components, differences and advantages. We then analyzed the use of existing patterns by Android developers community to understand the primary difficulties met by developers while choosing an architectural solution. We integrated the EUTECH SSII Android development team to observe solutions that was already applied to existing applications and interviewed developers about architectures and how are they used in the Android development process. We also searched for Android architecture-related questions on StackOverflow - the largest community of software developers collaboratively answering software-related questions - and blog posts to come up with a set of solutions and their disadvantages. Section 5.3 presents the obtained results of this phase.

Based on the obtained data, we proposed an architectural design pattern that is presented in the section 5.4.1. The pattern effectiveness was then evaluated through multiple examples and prototypes and was adjusted through multiple iterations. One of the last applications prototype is described in section 5.5. The model was then applied to real application development within EUTECH SSII and again reevaluated and readjusted through multiple iteration. Typical situations and special cases were also identifies in this phase that resulted as a set of implementation patterns. The implementation patterns are presented in the section 5.4.2 and special cases are added into the discussion section.

Several evaluation method were applied to the proposed architecture. The prototyping phase permits validating the solution by scenarios for maintainability, reusability and extensibility: most common code quality cirterias. Real application implementation shows the applicability of the architecture to real applications. Redevelopment of one existent application using the proposed architecture allowed to obtain the code quality metrics and measure the gain. Finally, the feedback from long architecture usage by Android

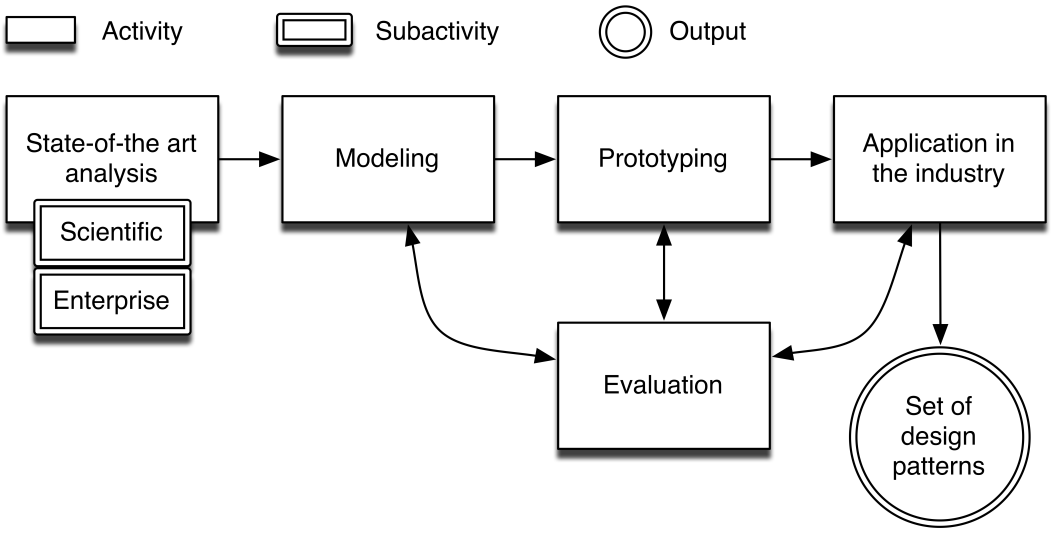

Figure 5.2.1: Research methodology for Android passive MVC

developers showed the long term benefits of such pattern and the applicability of the architecture to development of different applications.

5.3 Developers' experience and difficulties

Activity causes major difficulties in implementing the known architecture: is it a View, a Controller, a Presenter or none of them?

The most common way to develop Android application is to create one Activity per screen. Naturally, Activity initialize Views and intercepts actions made on Views by the user (methods corresponding to actions could be directly defined in layout.xml, Activity should implement the defined methods). Presentation logic of the full screen and a communication with the core of an application is often situated in the Activity making it very heavy and complex [110]. Thereby Activity managing actions and the presentation logic of the full screen behaves as a View-Presenter couple of MVP or a big Controller of MVC. The simple schema is shown in Figure 5.3.1.

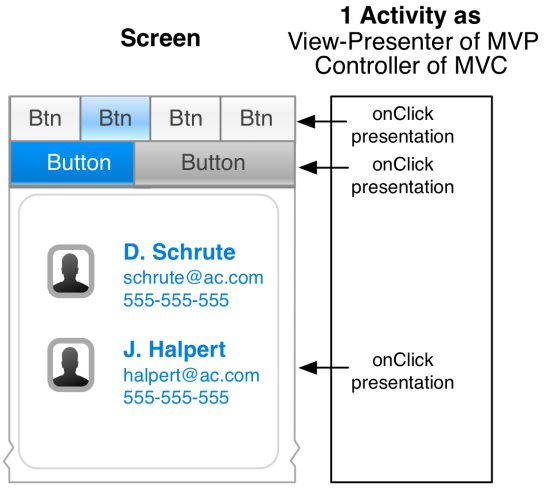

Figure 5.3.1: Activity as 'View-Presenter' of MVP or a 'Controller' of MVC

We also found examples where a single Activity manages an entire application: all possible actions of an application and the full presentation logic is managed by only one class - Activity.

The View-Presenter or thick Controller implementation leads to multiple problems: reutilization, maintenance, extensibility, code clarity, team development and even performance. Parts of code integrated into single Activity cannot be reused, methods can only be copied to another Activity making the redundant code. Any additional View and action complicates the Activity. A modification of one action repeating on several screens requires modification in all related Activities (assuming one Activity per screen). Activity can contain the implementation of very different actions non related to each other, this can make the Activity very complex, unreadable and incomprehensible. Activity is kept in memory while the application is running, thereby a very big Activity affects performance. Finally, the modification in the user interface and application logic can lead to the need of full redevelopment of all Activities.

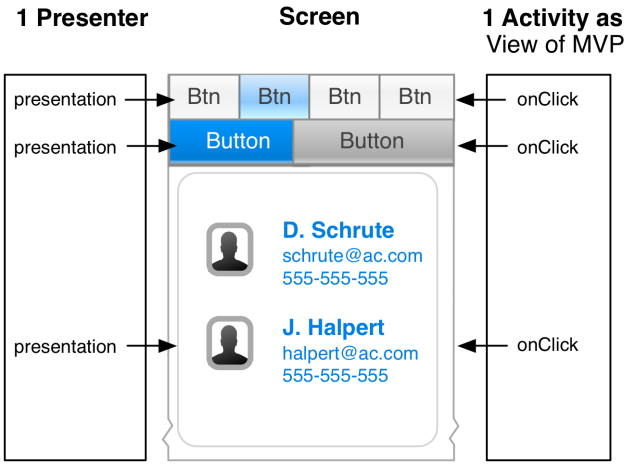

Figure 5.3.2: Activity as 'View' of MVP with additional Presenter component

Some developers improve the architecture placing the Activity as a MVP View and put the presentation logic to the Presenter component. It makes Activity lighter as it manages only actions available on one screen, but reutilization and maintenance problems remain the same as explained above. The simple schema is shown in Figure 5.3.2.

Another MVC implementation place Activity instead of View and creates the Controller separately. Activity cannot be implemented as a View due to the particularity of the component, but Activity can initiate and regroup all Views on the screen. Thereby we obtain very thin Activity and thick Controller handling all screen events and managing the presentation logic. The simple schema is shown on Figure 5.3.3.

Even if Controllers could be reused by other Activities the full object is needed to reuse methods related to one View from the previous screen; the structure of application becomes unclear due to the reutilization. Problems of extensibility and maintenance persist.

This solution works for simple applications where one Activity represents one visual block, while Activity usually manages several Views: main screen, menu, dialogue box, lists, forms, etc. In complex visual applications Controllers becomes heavy.

Assuming the Activity cannot be a View, as Views are already available and extensible on Android, few developers replace the MVP Presenter with Activity. The simple schema is shown in Figure 5.3.4.

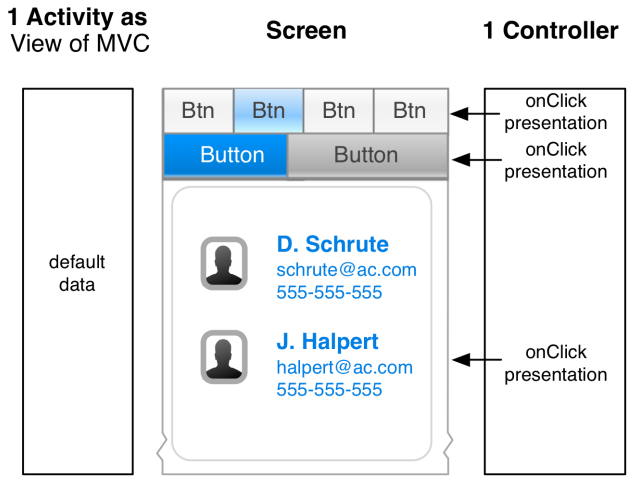

Figure 5.3.3: Activity as 'View' of MVC with additional Controller component.

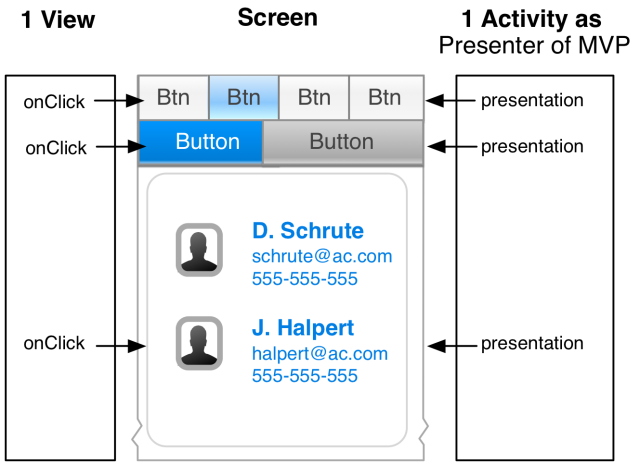

Figure 5.3.4: Activity as 'Presenter' of MVP with additional View component.

This solution makes View intercept event of all visual components available in the screen; presentation logic moves to Activity, but similar problems appear: reusability, extensibility, code clarity, etc. Presentation logic cannot be reused, but should be copied to another Activity if needed. The complexity of a single View increases with the number of events. This is suitable only for very simple applications with very simple screens.

The appearance of Fragments could have solved the architecture ambiguity, but Google proposes new components without suitable documentation about the utilisation of Fragments, thereby creating new ambiguity instead of solving the problem. Now developers ask themselves in what cases they should use the Fragments and not the simple Activity, what component should handle actions and presentation logic, where to place the Fragment management code, etc. We find previously explained MVC/MVP solutions, where the component that is not implemented as Activity becomes a Fragment (e.g., MVP implementation where Presenter is implemented as Activity and View is implemented as Fragment).

Nowadays more and more developers use Fragments, often as Controllers of MVC, but questions about presentation logic, communication between components, the actual purpose of components and its logic remains unanswered.

The full code organisation needs to be clarified: what existing component should be used and for what purpose, what type of code can be placed in those components and when and for what purpose should additional components be created? We find many applications where the part of core logic of an application is placed in the Activity or in the Controller/Presenter making them even more complex. Developers are often unsure about the decomposition of an application to Activities and Fragments and have problems in core organisation.

Note that over the research, examples developed using HMVC or PAC architectures were not found.

5.4 Android Passive MVC

This section gives the theoretical background on 'Android Passive MVC'. It presents the architectural pattern as well as its vocabulary and also dives more details on special-cases patterns helping to implement 'Android Passive MVC' with Android specific components - Fragments.

5.4.1 Presentation

The MVC model is taken as a base for the proposed architecture, as MVC is well-known and widely used in desktop and web systems as well as in iOS mobile development. Developers coming from other systems would be able to easily appropriate the Android development architecture.

Activity is an inevitable component of the Android application. Previous experience of the Android community shows Activity does not fit well on the MVC model, while it seems to be well adapted to developers' needs. Many View components are already available on Android but Activity cannot be a Controller or a Model. From the previously described development experiences one can see that the screen cannot be represented entirely by one or two components. We observe that the screen should be decomposed into many logical parts and each part should have the related components. For the proposed

'Android Passive MVC' architecture, the MVC triads are created around Activity making the Activity the fourth component.

Activity can be seen seen as a main screen (parent) controller in HMVC model. The simple schema is shown in Figure 5.4.1.

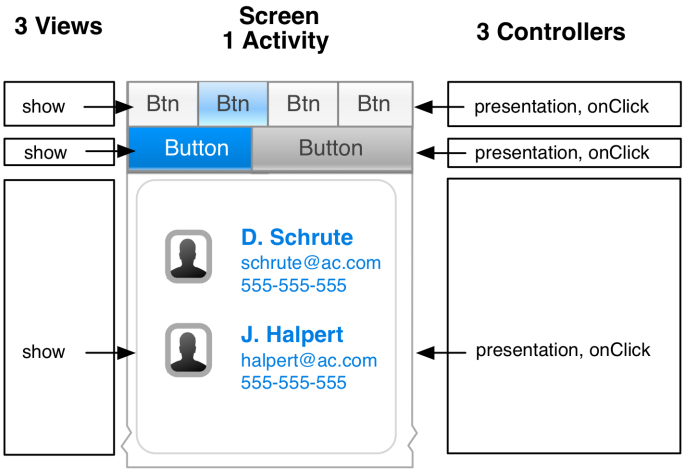

Figure 5.4.1: Activity as an intermediate component between Views and Controllers.

An observer-observable pattern is relevant for multi-screen systems but only one screen is active at a time in Android applications. This pattern implies keeping in memory Views and Models that appear heavy for the mobile environment, therefore the Passive Model MVC was chosen as a basis for the architecture.

In the 'Android Passive MVC', Activity becomes an intermediate component between the Views and the Controllers. The Activity represents a screen controller or, in some cases, a main controller for a group of logically conjoint screens.

Controllers take the event handling responsibility and the presentation logic making the Activity lightweight. Controllers are also lightweight because one Activity can interact with many small reusable Controllers. Controller handles events and presentation logic only for a small number of views logically linked together. Controllers should not contain any code related to the core application functionalities.

The Views are the interface components, such as a form, a menu or a list of elements. View components contain methods that allow the setting or obtaining of data from the user interface on Controller demand, the setting of event listeners on visual components and the modification of visual components (set errors, change colours, etc.). Views are created if necessary extending Android predefined Views, otherwise the Android predefined Views can be used directly. Views are independent and do not communicate. Views should not contain any application logic or data.

The Model in this architecture is a Domain Model containing the application core logic and data. The simple scheme of the Android Passive MVC architecture with all components is shown in Figure 5.4.2.

The starting Activity creates a link between a View and a corresponding Controller to make them communicate directly. Controller set up the View it is responsible of: visual presentation and the data. The Controller handles events from the user action (e.g., button click), calls necessary methods from the Model and then updates the View on Model response.

Simple hierarchy of Activity and Controllers depending on this Activity will be suitable for many simple applications, although Android interface is a modular interface similar

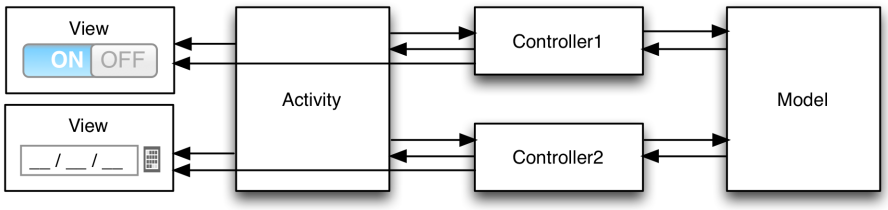

Figure 5.4.2: Android Passive MVC.

to Java. The propose architecture to organises View-Controller couples in Hierarchy as HMVC and PAC architectures. The actions of interface modules controlling another interface module will be organised as parent-child controllers.

We define two type of controller: Mediate Controller and Coordinating Controllers. The names are borrowed from iOS architecture also having two types of controllers.

Coordinating Controller is a simple Controller for an independent part of the screen coordinating the presentation logic and events of its Views. This Controller can call the Model, modify its View visualisation, show dialogues, call Activities but does not exchange Controllers. The Coordinating Controllers do not have any child controllers. Coordinating Controllers are very reusable and make an application very modular. Coordinating controllers can be perceived as low-level PAC agents.

Mediate Controller often corresponds to the part of the interface modifying or exchanging Coordinating Controllers (part of the interface). Menus in the interface would often correspond to the Mediate Controller. Mediate controllers can also initialise child Mediate Controllers (e.g., for a submenu). Activity Mediate Controller manages Activity replacement. Mediate controllers are similar to the Intermediate-level PAC agents with the difference that they have direct access to the Domain Model (application core).

Mediate Controllers are not very reusable as they need all their children to function, although Mediate Controllers show the presentation logic of the application; the logic of the interface can be modified by updating or changing the Mediator controller.

To keep components loosely coupled it is recommended to ensure communication between Controllers and Activity via interfaces. The communication schema is shown in Figure 5.4.3.

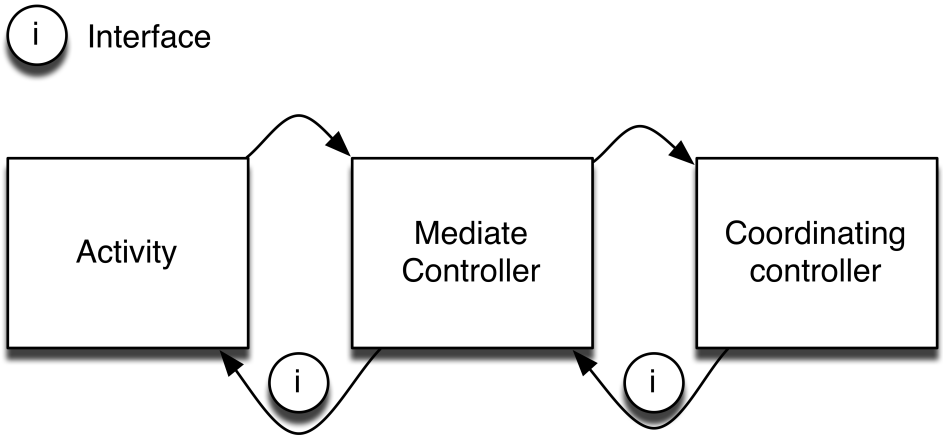

Figure 5.4.3: Communication between Controllers.

Android Passive MVC makes Activity lightweight by moving all event handlers and

presentation logic to Controllers and interface management to Views. Views and Controllers created on demand avoid unnecessary objects, saving memory. Android predefined View fits the model and new Views could be created by developer are reusable in future applications. Coordinating controllers are very reusable and makes the application very modular. Mediate Controllers are less reusable but enable easy modification of the logic of the application only by modifying the Mediate Controllers.

Developers can easily modify or remove application components by only updating or deleting the corresponding View-Controller couple. Application can be extended with View-Controller couples. The Model is independent from the View, the Controller and the Activity. The user interface could be replaced without any impact on Model, therefore the maintainability of the application is high.

5.4.2 Implementation

This section presents some examples of Android Passive MVC implementation. It introduces more details and special cases of architecture usage. Controllers of AP-MVC can be implemented with simple Java classes or with the Android Fragment component.

Both implementations are suitable for the new manually created Activities. Some predefined Activities, especially from third-party libraries, will possibly not fit the implementation.

5.4.2.1 Fragments usage

Fragments is an Activity-like component that can represent and control a part of the interface. Fragments can be used to implement Controllers in Android Passive MVC. Since the introduction of Fragments, Google insists on the high usage and integration of Fragments into Android applications and deprecates Activity-based functionalities. Fragments propose multiple advantages in Android Controller implementation versus simple Java classes:

- Fragments are native Android components automatically linked to the Activity via layout.xml having the native possibility to communicate with the Activity.
- Fragments have their life cycle linked to the Activity.
- Fragments are automatically linked to Views via layout.xml and can retrieve Views to communicate directly.
- Android integrates the Fragment manager: Fragments can be easily replaced, deleted or added to the Activity.
- Activity has access to all attached Fragments.
- Fragments integrate the back button gesture: option of saving the Fragment with its state in the back stack and retrieving it on back button press. One can also choose to retrieve the existing Fragment with its state or create a new Fragment with the default state.
- Fragments can manage other Fragments.

A Fragment is created for each piece of an interface having an action or several logically linked actions. All actions should be distributed between Fragments and should not be added directly to the Activity. Even for only one simple form (e.g., login form) the Android Passive MVC imposes the use of the Controller (Fragment) along with the Activity. This makes the application more modular and improves maintainability, the same independent Fragment can be easily reused in the future. Figure 5.4.4 shows a single Activity with a single Fragment and a single Activity with two independent fragments.

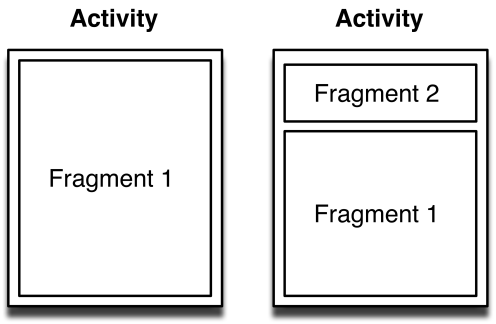

Figure 5.4.4: AP-MVC impose the creation of Fragment event if the only one is currently used within Activity

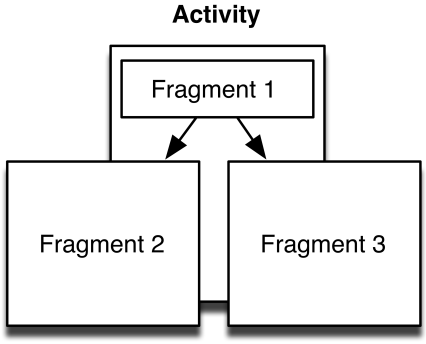

Figure 5.4.5: Mediate Fragment corresponding to the possible menu that exchange 2 Fragments depending on intercepted action

Fragment is linked to corresponding Views via the layout.xml. Fragment should not retrieve other Views available in the Activity to stay independent.

Fragment can play the role of Mediate Controllers and manage other Fragments or change Activity. One Fragment cannot exchange itself with another Fragment therefore it needs a parent Fragment (Mediate Controller) to perform the transaction. Figure 5.4.5 illustrates an example.

Android gives the option of adding a Fragment that is not directly linked to the interface, permitting the creation of Mediate Controllers without visual components. In some cases, actions from different screens and different Activities can be combined in one single Mediate Controller if those screens are logically linked. For example, a form can have several pages (screens) with 'next' button or 'go to first page' button; the appearance of the screen changes on click event interception. In this case, rather than adding an action to each fragment separately, making them dependent, the developer should create a Mediate Controller combining those actions in one place. Figure 5.4.6 illustrates this example. Therefore, in the case of user interface reorganisation (e.g., add new screen in the middle of the chain) only the Mediate Controller needs to be modified. The same should be done for a bundle of dependent fragments within a single Activity.

Fragment initialises itself with default data or the data recovered from the bundle (Android mechanism to pass the data between Activities), therefore Fragments stay maximally independent from other Fragments. Some Fragments can be initialised by an Activity or parent Fragment (Mediate Controller) to increase reusability. The possible communication between Fragments and Activities is shown on Figure 5.4.7 with two Mediate Controllers and one (the upper right) Coordinating Controller. Fragments should rarely

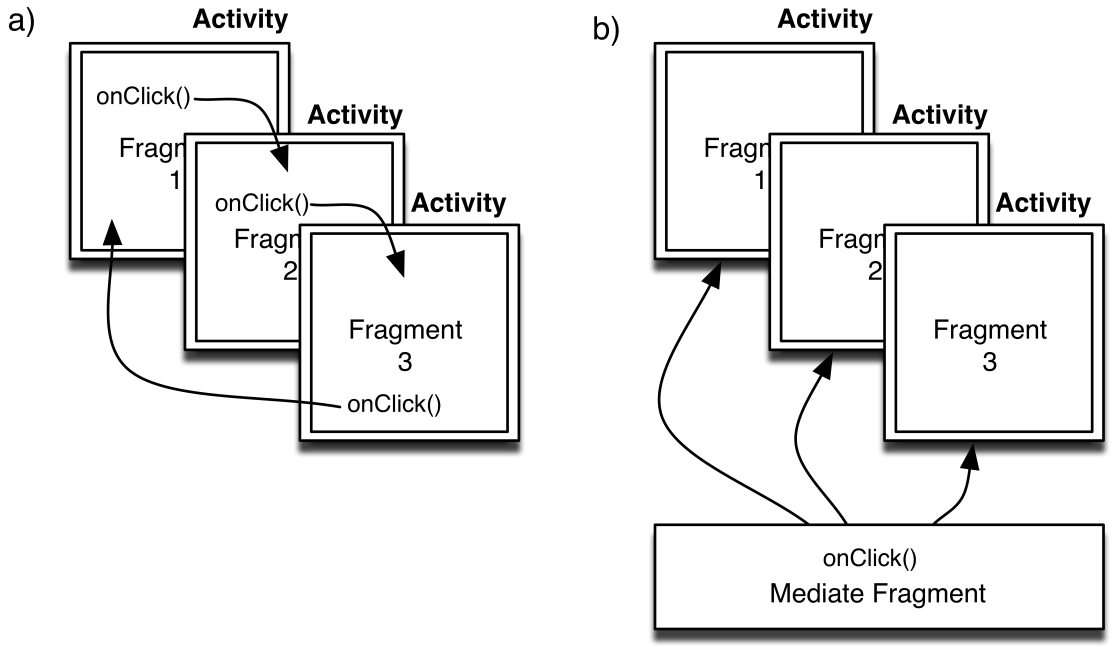

Figure 5.4.6: A chain of dependent Fragments or Activities. a) Direct calls make dependent Fragments b) Mediate Controller makes components independent

have a callback to parent Fragments but if necessary the callback can be implemented with interfaces.

Developer should avoid high hierarchy between Fragments within a single Activity as parent Fragment is linked to the child Fragments. Activities make components more independent and simplifies the Fragment management.

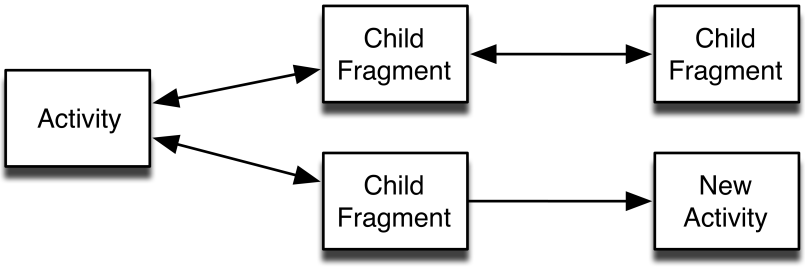

Figure 5.4.7: Communication between Fragments and Activity.

In some cases, Fragments depend on each other (cannot be a parent-child, but should initialise each other) - the circular dependency between Fragments is observed. Figure 5.4.8 shows an example: a list of folders and a file path to the parent folder. By clicking on the folder in the file path, the folder list should be updated; by clicking on the folder in the list, the file path should be updated. Another example is a mobile tablet with a large screen that contains the statistics data represented in different Views: tables or graphs. Changes in any of the Views should affect all other Views.

This is also a typical case where the Classic MVC is very pertinent where the Observer-Observable pattern can be used instead of Mediate Controller: several Views represent the data using the same Model, Controllers can modify the Model and all Views should

be updated. Although Mediate Controller keeps components more independent.

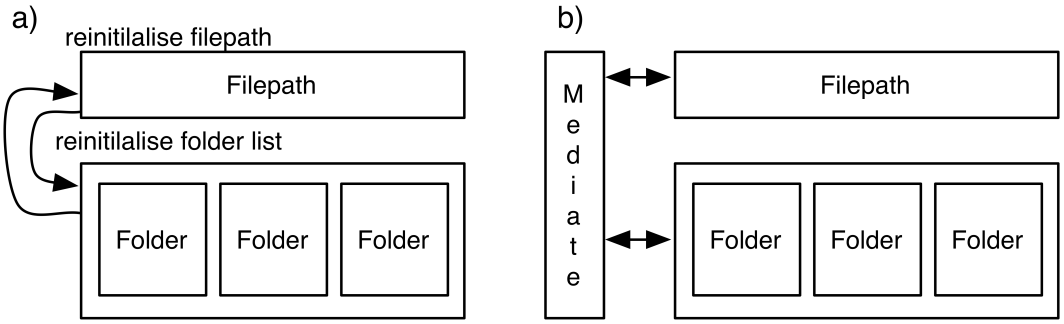

Figure 5.4.8: Circular dependency between Fragments a) Direct calls, dependent Fragments b) Mediator Controller makes Fragments independent

It is possible in Android to retrieve one Fragment from another Fragment and to call the initialisation method. Although this makes very tight coupled components. A better way is to make those Fragments communicate via listeners implemented by a parent component: a Fragment playing the role of Mediating Controller.

5.4.2.2 Java classes

Controllers can be implemented as simple Java classes, the same as Views. Controller should be linked to the Activity, therefore the Activity should implement a Controller listener interface and pass it to the Controller to establish the communication. The components communicating via listeners are loosely coupled and the communication of Android components via interfaces is presented in [36].

As the Activity would initialise the Controller, it can communicate with Controller directly, but the communication via interface is preferable. Activity should also retrieve the View and pass this View to the Controller to establish a direct communication. The communication between the Controller and the Model should be established via listeners (interfaces).

Figure 5.4.9 shows the Android Passive MVC implementation diagram. Listeners increase the performance of the application and create a weak coupling between components that improve maintainability.

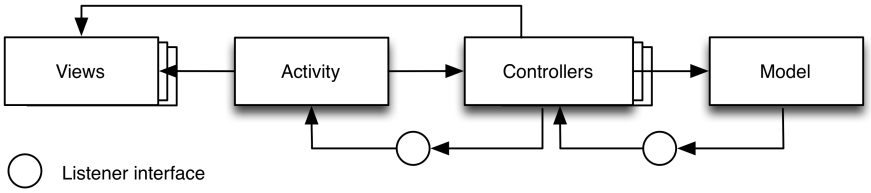

Figure 5.4.9: Android Passive MVC implementation

A login screen with a classic login form to enter the login and password represents an example of architecture implementation without Fragments; if the login is successful the user goes to the welcome page, otherwise an error message appears.

The example contains two Activities: Login Activity managing the login page and Welcome Activity for the welcome page. The login form is managed by Login View and

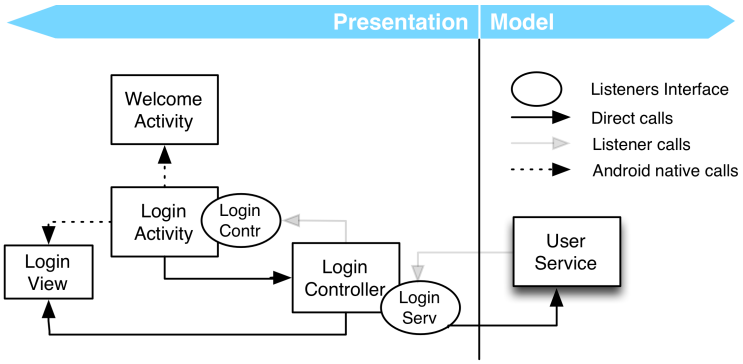

Figure 5.4.10: Login implementation example

Login Controller. Login Activity implements the LoginControllerListener interface to be able to receive calls from the Login Controller. The schema is shown in Figure 5.4.10.

Login View contains methods for obtaining login and password (getters), methods to set button listener and methods to set errors. Login Controller handles events from the login form implementing the onClickListener; while the button is pressed, Controller calls the model that launches simple verifications. If login is successful, Controller opens a welcome screen. If login fails Controller sets up an error message.

5.5 'Tweetle' Android application and Android Passive MVC

This section illustrates the implementation mechanism of 'Android Passive MVC' on the prototype: Twitter client (microblogging social network) application named 'Tweetle'. 'Tweetle' allows users to see the messages that he sent to Twitter himself and the messages of two types of friends from twitter: messages from profilyser that user reads (followees) and messages from profilyser that read the user (followers). User can tweet using the dedicated screen and also 'retweet' any message from any list.

Application contains three screens, one mail menu with three buttons; one of the screens has an additional submenu corresponding to followees and followers. The user can see the Twitter timeline, send tweets and visualise the list of tweets of his followers and followees. The bar with the copyright button showing the application author's name appears permanently on the up of the screen. The interface of 'Tweetle' is depicted in Figure 5.5.1.

One can see that all three screens are logically linked together by the main menu; one screen is divided into two logically linked parts by the submenu. Main actions are clicks on the main menu and clicks on the submenu. Additionally, by clicking on any of the list, a user can retweet the message. Finally, the button sending the tweet is presented on the last screen.

One can notice that two Mediate Controllers are required for the main menu and a submenu and at least two Coordinating Controllers for the copyright bar and the list of tweets.

Application can be implemented in two ways. Both implementations are presented and advantages and disadvantages of each are discuss.

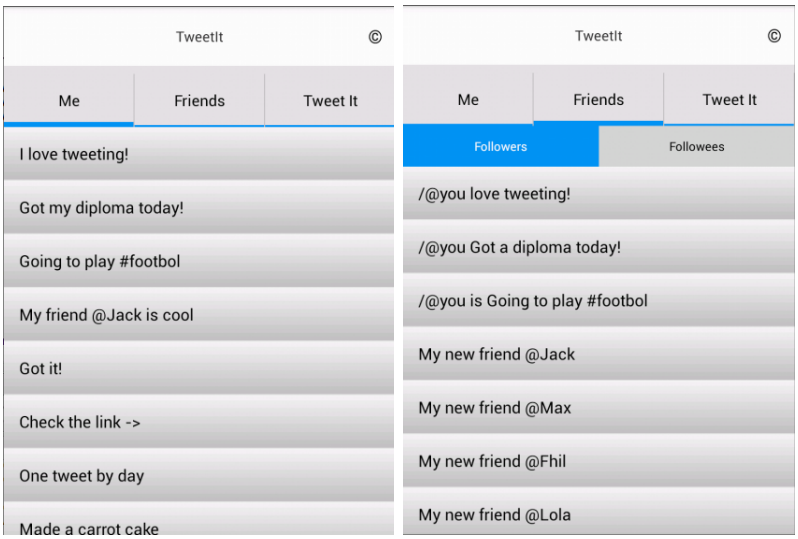

Figure 5.5.1: 'Tweetle' application user interface

5.5.1 Fragment mediate Activities

Each different screen interface is managed by a separate Activity. This possibility is similar to "before fragments appears" implementation solutions. In the example, the main menu imposes three Activities with three buttons. One can also suppose to create one Activity by submenu or to keep the single Activity for both submenu actions as was done in the implementation example as modifications on the screen are minimal.

The screen can be decomposed into four Fragments: mostly repetitive elements on the screen. Fragment should only contain the presentation logic and actions that are logically linked together. Different actions like "retweet" and "onMenuPressed" need different Fragments.

- The bar containing the copyright button corresponds to a copyright Coordinating Controller (Fragment). This controller is highly reusable even for different applications of the same developer. The bar and the button can be personalised with an layout.xml, but the Controller containing copyright action calling the dialog or a new Activity can be reused exactly in the same way in another application.

- The second Fragment is a main menu Mediating Controller. This controller will change the screen (Activity) depending on the button pressed. Controller also manages the presentation of the main menu: active and non-active buttons.

- The third Fragment is a list Fragment: retweet Coordinating Controller. The same Fragment can be used for all lists as the user action is the same for all lists of the application and there is no presentation logic.

- The fourth Fragment corresponds to the submenu Mediating Controller and manages the changes in the data of the list (reinitialise the data or change the Fragment) and the presentation logic of active and non-active buttons.

Last Fragment corresponds to the form permitting to send the tweet - tweet Coordinating Controller.

Activity plays the role of an initialiser of child Controllers or a main Mediating Controller. Mediating Controllers can also initialise themselves using the data from the bundle

to be more independent. Copyright Fragment is attached only by the layout.xml and do not need any additional initialisation. Activity initialise the main menu: call the Fragment method to set up active button and event listeners. Activity also initialises the list of tweets of the first screen: Activity as a main Controller can call the Model to retrieve the data and to set it to the list Coordinating Controller. List Coordinating Controller (Fragment) can retrieve the data itself either. For the Followers/Followees screen the submenu Mediate Controller (Fragment) with its default state is attached automatically to the Activity. Submenu Fragment initialises the list of tweets. Initialisation calls are depicted in Figure 5.5.2.

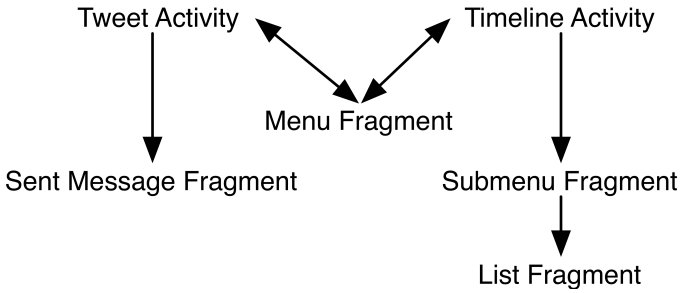

Figure 5.5.2: Activity by screen initialisation calls

5.5.2 Fragment/Activity mediate Fragments

Activity is created for a group of logically linked screen thereby only one Activity is needed for the current example. Fragments remains the same: one submenu Fragment, one list Fragment and 'send message form' Fragment. Menu actions can be implemented either in the Activity or a Mediator Controller (Fragment). The main menu should better stay independent in the Fragment instead of being in the Activity to enforce maintainability. Main menu Fragment manages clicks on buttons, apply visual modifications on buttons and exchanges other visible Fragments. The submenu Fragment have an child list fragment to manage: the information shown by the list depends on the action made on the submenu. Initialisation call schema is depicted in Figure 5.5.3.

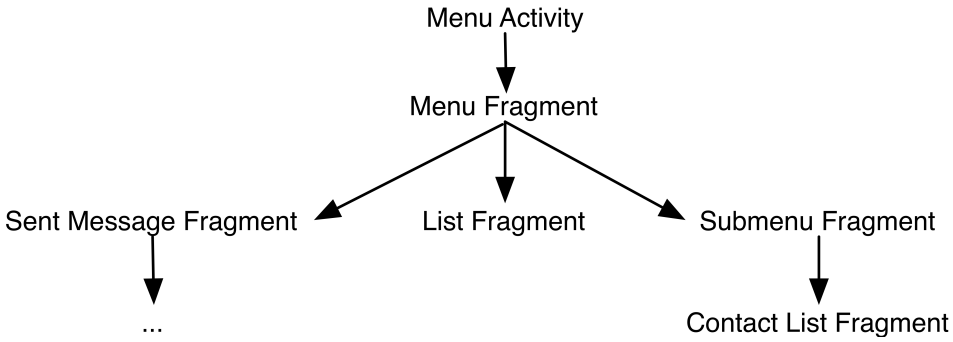

Figure 5.5.3: Activity as Main Menu

5.5.3 Advantages and disadvantages

Both implementations are very similar but have advantages and disadvantages.

The first implementation is easy to set up and keeps the structure clear. Activities are nearly empty thereby the only active fragments take place in memory, the number of fragments is also limited and easy to manage. 'Back' button is managed automatically. Android integrates a bundle mechanism allowing information to pass between Activities; Fragments could initialise themselves retrieving the information from the bundle while the Activity is changed. Otherwise, this implementation is only suitable for lightweight interfaces as all Fragments are reinitialised for each screen. The time response increases significantly if heavy images appear on the interface.

The second implementation has a clear structure but could be trickier to manage. This solution permits to reinitialise only necessary fragments, therefore can be used with more heavy static images, for example with the background image. Although, developer should assure to keep in memory only visible fragments. A large number of Fragments managed by a single Activity can be complicated and heavy if all Fragments are kept in memory. Fragment should be manually added to the back stack to manage the 'back' button. The second implementation is also useful for Activities aimed at being shared and at returning messages to other applications: this type of functionality should be implemented within a single Activity that another application could call for result.

5.6 Android Domain Model

The clear separation of presentation and business logic cannot ensure the application of good quality alone. The core of the application should also be implemented through patterns and gold architectures. Android application business logic structure is similar to any Java application core logic, therefore all patterns that can be applied to Java could also be applicable to the Android Domain Model, although the difficulties in Android Domain Model organisation are observed in the community.

This section goes further and gives some guidelines of the business logic of the application – the Model. Android applications have similar needs: internal database management and access, web service access and reusable components use. Clear main architecture of business logic is necessary to obtain the application of quality.

The Model of Android Passive MVC is a Domain Model containing business methods, web service call methods, database access objects, reusable methods and data model objects.

A Domain Model architecture should include components that are usual for Android applications, such as Database manager, Web services manager and Business logic. Those components should be independent, as the architecture should be adaptable. Reusable components should be also separated. The basic model architecture is shown in Figure 5.6.1.

The architecture of Domain Model proposed in this document is inspired by 3-tier architecture that separates the presentation, the business and the data access layers [111].

The business layer of the model regroups objects and methods that use web services, business services and reusable tools. Business services contain business logic. If an application works via Internet as well as locally, all necessary verifications are done in Business services, which calls corresponding methods. The communication between a presentation and a domain model layer are made via Business services.

The data layer contains Models, Data Access Objects (DAO) and Database Manager. DAO and Model are the implementation of the Data Access Object pattern. Model contains data being persisted in the database or retrieved by web services calls. Model

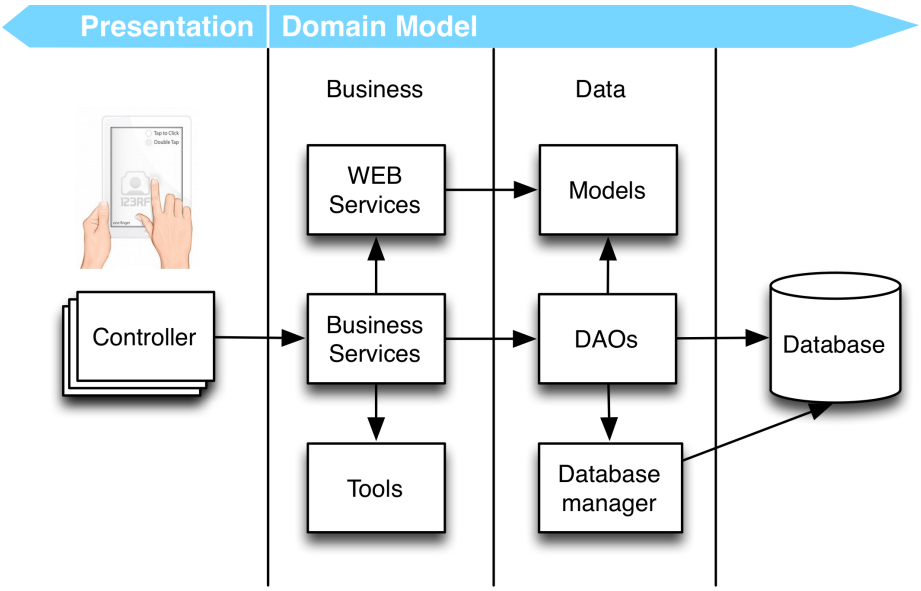

Figure 5.6.1: Domain Model Architecture

is a simple Plain Old Java Object (POJO) that contains only variables and their getter and setter methods. To avoid transcription of the Android Native Cursor object to Model objects, Model can encapsulate the Cursor object proposing getters and setters for a concrete value type available in Cursor. Data is manipulated and transferred through the application using those lightweight objects that are often called Data Transfer Object (DTO).

Persistence methods are organized in DAOs. DAO contains methods that enable the data in a database to be saved, deleted, updated and retrieved. Even if Android proposes an abstraction on the data access level with Content Provider, DAO simplifies the code of the application. The DAO design pattern creates a weak coupling between components and uses a Model object instead of an Android Cursor object in the application. DAO can also be used for the data stored in XML or text files. Good practice is to make DAO accessible via interfaces. It allows DAO modification (for example the change of SQLite to XML storage) without any change in Business services, which increases maintainability.

Database manager is in charge of database creation. Database manager exists only if SQlite database is used by the application. It stores the name of the database, and of its tables and methods to be able to create, drop, open and close the database.

This architecture regroups logically similar methods together, increasing cohesion. High cohesion facilitates the maintainability of the software. The final code of the application could be organized in packages by architectural component: Activities, Views, Controllers, Business Services, Tools, Web Services, Model, DAOs and Database. It gives the clear structure of an application and limits the package number. Additional packages could be created for interfaces, parsers (e.g., XML, JSON) and constants.

5.7 Architecture evaluation

The implementation of the model should improve the application and code quality: reduce the complexity of an application, clarify the code and improve extensibility. The coupling between components should be weak to avoid the modification of other components if one

is modified. Modules should be reusable [112, 17]. A smartphone has a limited memory, therefore the creation of unnecessary objects should be avoided. Objects remaining in the memory should be lightweight [28]. Modification in user interface or in navigation logic should involve the minimum modification of the application.

We evaluate the architecture in two steps. First subsection ensure that the architecture fits the lists of code quality criteria proposed by [112, 28]. Second subsection propose modification scenarios that can be applied to the 'Tweetle' and discuss the impact of each scenario on the implementation. Third, an experienced Android developer rewrites one of his latest applications using Android Passive MVC, compare results and give feedback regarding the model. Finally, two developers use the architecture for 10 months in their real life projects and give their feedback.

5.7.1 Code quality requirements

The evaluation of the architecture is based on the following three code quality evaluation criteria: maintainability, extensibility and reusability.

- Maintainability: option of modifying the system.
- Extensibility: option of adding new functionalities to the system.
- Reusability: option of reusing the same components in different functionalities of the system or in different systems.

The use of standard platform techniques is important for the model: the support of third-party functionalities could be interrupted making implementation of the model impossible. The Android Passive MVC could be implemented using Android SDK without any additional libraries.

A high-quality application has high maintainability and extensibility: codes have weak coupling between components, easy code suppression possibility and high testability. The Passive MVC architecture ensures high maintainability. Clear separation between presentation and business logic simplifies testability of components. Weak coupling between all layers is carried out via listeners. One component (ex. interface, DAO, web service) could be replaced or modified without changes in others. The extension or modification of the user interface itself is done by simply adding, deleting or modifying the view-controller couples.

The reusability of components make the code clearer and boost development time. The view-controller components of the Android MVC model could be reused through the application and could be easily embedded in other Android applications made with Android Passive MVC.

Good performance is especially important in mobile environments: resource utilization should be limited as mobile devices have little memory. Short response time is essential for modern users. The Android MVC architecture makes a very lightweight Activity component. Controllers, View and Model objects are also small and kept in memory only if used, which minimizes resource utilization. The use of listeners also slightly increases response speed.

5.7.2 Scenario-based evaluation

The scenario-based software architecture evaluation method is chosen to validate Android Passive MVC; the overview of such methods can be found in [113]. Scenarios enable eval-

uation of the architecture of a specific system and comparison of several architectures of the same system regarding modifiability. Scenario-base evaluation are applied to previously described implementations of Android Passive MVC to show the benefits of this architecture. Most scenario-based methods involve shareholders, software designers and an evaluation team for the real project to define possible modification scenarios and the ability of the architecture to support those modifications. The two architectures of 'Tweetle' that are described in the Section 5.5 and the most likely modification scenarios for the implementation are defined below.

1. Adapt the phone version to the tablet
2. Add new tab to the main menu
3. Move the main menu to the separate independent screen
4. Modify the appearance of the list
5. Add a bar containing the name of the active tab

The impact of each scenario on the both implementations is analysed and explained. Table 5.1 presents the quality criteria evaluated for each scenario.

Scenario #	Maintainability	Reusability	Extensibility
1	x	x	
2	x		x
3	x		
4	x		
5			x

Tableau 5.1: Evaluation criteria by scenario

5.7.2.1 Scenario 1: adapt the phone version to the tablet

'Tweetle' is an application dedicated to the smartphone usage, but can be adapted to smart tablet. Tablets in landscape mode have enough space to keep all three screens visible at one time, therefore the main menu becomes just an indicative name menu to define each list without any action. Tablet in portrait mode will have a mobile application behaviour.

The adaptation can be easily achieved with Android Passive MVC. Application should only be adapted for the tablet landscape mode. The developer should define a new layout.xml for the new tablet appearance: the new layout mainly consists of combining the existing layouts into one. Developers do not need to define a Controller (Fragment) for the main menu as there is neither action nor presentation logic needed. Other Controllers remain the same. Developers should add to the Activity a verification of whether the tablet landscape mode is active or not and set the corresponding layout. All Controllers defined in the layout.xml will be attached automatically. One can see that a very few modifications are needed to make the adaptable interface. This scenario shows the high maintainability and reusability allowed by the Android Passive MVC.

5.7.2.2 Scenario 2: add new tab to the main menu

It is very probable to add new tab to the existing menu. For example, 'Tweetle' need an extension with a map showing the newest geolocated tweets nearby. For both implementations, the developer should create a new map Fragment generating the map and Controlling actions on the map. Then, the developer can modify the main menu Fragment (controller) to add an action to the new button. For the first implementation the developer should also add a new activity-initialising fragment and an active button. Domain Model would be enriched with several new components as a new web service recovering geolocated tweets or a new DAO method recovering geolocated tweets from existing database have to be used.

One can see that only one Controller should be modified for this extension and several independent components are created. The modification of existing components is very light in Domain Model, too.

5.7.2.3 Scenario 3: move the main menu to the separate screen

The client wants to change the style of the mobile application creating a Windows 8-style menu screen with big square buttons and icons taking up the full screen. For the first implementation, the developer should create a new layout for the main menu and attach it to the new activity with the exact same Controller. The developer needs to check other Activities corresponding to the menu tabs to delete the initialisation of the active button, as it is not used any more if the initialisation was made in Activity.

The second implementation requires greater modifications: Activity can take a Mediator Controller role and replace the main menu with another Fragment, as the Fragment cannot replace itself. The developer could also pass to the first implementation modifying entirely the main menu Controller and creating additional Activities reusing all other fragments.

This example shows that for maintenance reasons the developer should preferably choose different Activities for the independent screens, as in the first implementation. In spite of the common menu, all tabs are completely independent and could be arranged differently in the interface while the application evolves. Fragment Mediate Controllers are less reusable but as they are very small they can be reimplemented easily. This example shows that the architecture resists extensive visual modifications and most Controllers remain reusable.

5.7.2.4 Scenario 4: modify the appearance of the list

It is possible to improve the visualisation of messages: to add an avatar, nickname, and make different colours for different lists. This can be done easily for both implementations. The developer should create an adapter to adapt the Tweet object from the Domain Model to the new visualisation in the list. The developer should only modify the adapter in the list Controller to modify the visualisation of all Controllers. If different visualisations are needed for different lists, the developer can create different Controllers or different Adapters and set up the visualisation in parent controllers.

This example also shows the maintainability of an application made with Android Passive MVC and the reusability of components.

	Original	Android MVC	% Gain
Number of Packages	25	17	32
Number of Classes	393	275	30
Number of Functions	2186	1683	23
Avg CCN	2,30	1,87	19
Max CCN	110	30	73

Tableau 5.2: TaskProjectManager statistics: the difference between the original and the 'Android Passive MVC' implementation and the corresponding gain in metrics.

5.7.2.5 Scenario 5: add new interface element

A new name bar to the initial 'Tweetle' application should be added. The visual appearance of this bar is the same for all tabs; the name corresponds to the tab name. For the first implementation, the easiest way is to add this bar directly into the layout.xml without any modification in the code. This is not possible for the second implementation. If the developer adds the modification of the view of the name bar to the main menu, two interfaces becomes dependent and cannot be used separately. Main menu could also notify the Activity to set up the name bar, but in this case the bar is not reusable in other Activities. The most reusable way to carry out the second implementation is to create a Fragment for the name bar. The main menu could pass the data to initialise the name bar as it initialises the lists instead of manipulating the view directly.

This example shows how the first implementation has maintainability advantages over the second implementation as the parent Fragments implementing Fragment transactions could be trickier to manage in case of the interface modification, but child Fragments can always be reused. This example also shows that Fragments have advantages even for the interface having no action but presentation logic.

5.7.3 Evaluation by developers

An Android developer with three years' experience agreed to test the Android Passive MVC in real life projects. He chose to redevelop one of his latest applications which had become complex and hard to maintain, extend and test. The application is called 'TaskProjectManager' and it enables tasks to be assigned to different employees and to view the full calendar of tasks on the screen by day, week and month. The application also generates reports according to parameters.

The old version and the novel version of the application developed using the Android Passive MVC without Fragments were compared. In spite of the fact that developers produce slightly better code while redeveloping the same application due to greater experience, our measurements show the impact of Android Passive MVC on the redevelopment.

Measurements of both versions of the application are made with JavaNCSS [114], a source measurement suite for Java, and the results are shown in Table 5.2. Android Passive MVC reduces all code parameters.

For each comparison feature denoted i, the gain is calculated as the difference between the original and the Android Passive MVC applications scores (resp. $Original_i$ and $AndroidMVC_i$) divided by the original application score (i.e., $Original_i$).

$$Gain_i = \frac{Original_i - AndroidMVC_i}{Original_i} \tag{5.7.1}$$

where:
$Original_i$ - measurement of the feature i taken on the originally implemented application
$AndroidMVC_i$ - measurement of feature i taken on the Android Passive MVC implementation
i - the comparison feature

The Android Passive MVC helps with organizing classes in packages. The original version of the application had many packages created partly using the MVP model, partly with the application logic, and partly following the Android components' names. The limited number of packages of the Android Passive MVC version gives the application a clear structure dividing the Domain Model from the interface management.

The full code became smaller: both the number of classes and the number of functions were reduced. We observed the application had a main menu appearing while the calendar was visible. Calendar had different modes of functionality managed by different activities with a huge presentation method managing the appearance of different activities. Menu actions were found multiplied in those activities. The Android Passive MVC enables high reusability of components and structuration of presentation logic.

The code complexity is evaluated using Cyclomatic Complexity Number (CCN) [115]. 'Cyclomatic complexity measures the number of linearly independent paths through a program module' [18]. Normal method complexity without any risks is 1-10 CCN, with 11-20 CCN the complexity is moderate, with 21-50 CCN the complexity is very high and with CCNs greater than 50 the program is untestable. Table 5.2 shows that the average complexity of the application has decreased slightly. The maximum CCN dropped significantly: an original version has methods with CCNs of 40, 50 and even 100 and 110 mostly for Activities handling a huge number of events, while the new version has the only JSON parser with a CCN of 30 and several methods with a CCN of 10 to 15 in the application core.

The developer's feedback explained that the Android Passive MVC model is easy to understand and to follow. The final application was visibly more reactive: the response time became almost nil, while the users of the original version complained about a very long response time for each screen. The Android Passive MVC version is open to extensions and easily modifiable. The developer said that he had already added more functionalities to the new application before transmitting the code for the CCN analysis. Application components are not only reusable in the application, but could also be reused in future Android development.

The same developer and his colleague continued to use the Android Passive MVC in their everyday job of Android application development for clients. They have tested the version with Fragments and consider it even easier due to the many predefined Android actions.

We obtained a new feedback after 10 months of testing. The developers recognise the improvement of development process since the architecture was introduced: they were able to test both types of Fragment implementations but mostly the first one. The software design state became shorter. They were able to reuse components from one application in another with slight controller modification: photo gallery, search views, 3D and PDF visualizers, horizontal scroll view, etc. They reported the shorter development time due

to the clear structure defined by the architecture and easier group work. The division of projects on tasks became simpler and conflicts in code merges became less frequent.

They also noted that they were able to integrate updates rapidly while some old projects without the architecture were redeveloped entirely because of updates demanded by the client.

The developers also discovered that the architecture helps students in practical training to do better and gives them more autonomy: the company offered Android development practical training for two student during the experiment. In older project, students without experience needed continual supervision and without it produced little maintainable code: some applications that involved student needed a full redevelopment to suite new client's demands. Student, having a short practical training during the experimental Android Passive MVC usage, showed better results while having the same background as previous students involved into Android development.

The developers also noted that the architecture simplified the work with a colleague's code if he is absent, thanks to the common logic, naming and structure.

5.8 Discussion and future work

This section presents the possible extension of the 'Android Passive MVC' with concepts from MVP and AM-MVC.

5.8.1 Android and MVP

In the MVP, the View and the Presenter correspond to a full screen interface. It is not suitable for Android, as visual block embedded into one View could be reused on several screens. Although architecture similar to MVP can be implemented on Android around the Activity making triads for each visual piece (such as Android Passive MVC).

The Presenter manages the presentation logic of the View and communicates with the model. The difference between the implementation of both architectures is the event handling.

Events in MVP are handled by Views and the action is transmitted to the Presenter. Unlike the Android Passive MVC, in MVP View listens to the events and only calls corresponding methods in Presenter instead of letting the Controller listen to events directly. This implies the creation of additional classes for each View (visual component) implementing event listeners.

The code of Controller/Presenter remains the same. The only difference is that instead of one method handling an event (e.g., click event) Presenter uses many methods corresponding to the action to be carried out on one particular event (e.g., click on search button). This slightly reduces the Controller complexity and allows easier testing. The initial event handling method is placed in the View and remains very simple and does not need any tests.

The MVP implementation was not chosen as a base architecture as the implementation of two models is very similar. The existence of Controller having a very complex event handlers can justify the use of MVP instead of Android Passive MVC but it is a rare case. The MVP implementation can be considered as an extension of Android Passive MVC for the exceptional particular use.

5.8.2 Android and AM-MVC

The AM-MVC adds the Application Model (AM) component to triads moving the presentation logic to this component. This component can be used in Android Passive MVC in some cases where the action of visual component remains the same but the visual presentation changes. This kind of situation is visible on 'Tweetle' implementation with reusable lists. Instead of unambiguity as to whether the population of the list should be done in the Controller or in the parent Controller, different AM components could be created for each list then, depending on the screen, Controller can initialise the necessary AM object.

This situation was not found very common and the improvement sufficient to include this case directly into the Android Passive MVC. Similar to the MVP, the AM component is considered as a possible extension for the exceptional use.

5.9 Conclusion

The architecture plays an important role in the development of good quality applications. The Android developers were missing unified defined architecture that created a gap.

This study have analysed some well-known architectural design patterns and proposed an Android architecture solution based on an MVC and PAC/HMVC design pattern. This work also proposed the Domain Model organization for the Android application that helps to structure the core functionalities. The implementation examples for several common cases in Android development were provided as well as a concrete implementation of a Twitter client mobile application - 'Tweetle'.

The architecture defined can simplify the work of novice and experienced developers alike and enable the creation of less complex and well-structured applications.

The architecture was evaluated in several ways: scenario-based evaluation showed the high maintainability of the ex- ample implemented with Android Passive MVC. One existing Android application was reimplemented using Android Passive MVC, resulting in better maintainability, extensibility and performance. The complexity of the new implementation was considerably lower. Two developers were involved in long-term testing of the architecture on real projects and collected positive feedback on Android Passive MVC. The architecture explanation is available online to reach a larger population and to collect more feedback.

It is important to note, that Android Passive MVC could also be applied to other systems similar to HMVC and PAC architectures. It requires the main component (main Controller) implemented as Activity in Android but can be represented otherwise in another system.

This work was presented on The Fifth International Conferences on Pervasive Patterns and Applications PATTERNS 2013 [116] where obtained the best paper award. The extended version of the article was published in the International Journal On Advances in Software, volume 7 in 2014 [117].

Part III

Security and Privacy

Chapter 6

Detecting abusive applications: permission usage patterns for applications' classification and anomaly detection

Abstract

Android is one of the mobile market leaders, offering more than a million applications on Google Play store. Google checks the application for known malware, but applications abusively collecting users' data and requiring access to sensitive services not related to functionalities are still present on the market. A permission system is a user-centric security solution against abusive applications and malware that has been unsuccessful: users are incapable of understanding and judging the permissions required by each application and often ignore on-installation warnings. State-of-the-art shows that the current permission system is inappropriate for end-users. However, Android permission lists do provide information about the application's behaviour and may be suitable for automatic application analysis. Identifying key permissions for functionalities and expected permission requests can help leverage abnormal application behaviour and provide a simpler risk warning for users. Applications with similar functionalities are grouped into categories on Google Play and this work therefore analyses permission requests by category.

This study proposes a methodology to characterize normal behaviour for each category of applications, highlighting expected permission requests. The co-required permissions are modeled as a graph and the category patterns and central permissions are obtained using graph analysis metrics. The obtained patterns are evaluated by the performance of the application classification into categories. Finally, this study proposes a privacy score and risk warning threshold based on the best metrics. The efficiency of the proposed methodology was tested on a set of 9,512 applications collected from Google Play.

6.1 Introduction

Mobile applications are extensively used worldwide and new mobile applications are added every day to mobile markets - platforms for the distribution of mobile applications. Android - one of the market leaders - offers more than 1,5 million applications on the official Android application store named Google Play (June 2015).

The applications are grouped into 35 categories. When a developer adds an application to Google Play, he should choose the most appropriate category from a list of categories proposed by Google, such as weather, communication, heath and fitness, productivity, tools, etc. A search by category or by keywords is offered to users.

Google Play provides certain information helping users to decide which application to install: screenshots give an interface overview; users' ratings and comments reflect the stability and usability of an application. Although users are looking for attractive and useful applications with no bugs, they should take into account other factors before installing an application such as security and privacy, especially in the context of BYOD (Bring Your Own Device). Furthermore, previous studies such as [45] show that users are concerned about privacy and security and want to have applications with greater respect for privacy.

Android applications can be written by any developer and do not require any certification or validation before being made available on the store. As a result, poor, malicious or applications abusively requiring and collecting personal data coexist with benign Android applications on Google Play [118, 94, 96, 50, 60]. Google Bouncer[1] now checks applications for malicious code, but no validation takes place for applications that abusively require permissions and collect user data, even when it is unnecessary for the proposed service.

One of the Android platform security mechanisms is the permission system. This list is supposed to warn users about hazardous and abusive applications, but, unfortunately, permission lists have been shown to be ineffective for this purpose. First, users see permission lists as a repeated warning or a license agreement that must be accepted to obtain a service. Permission lists are only shown in the final step before installation when other criteria for the user's decision have been met, and therefore the permission list is considered an obligation rather than a decision factor [46, 45]. Second, users often do not have enough background to understand the meaning of permissions and their possible harm. Third, permissions are shown entirely out of the context, which prevents the user from understanding their purpose. Finally, some permissions are so frequently required that users do not pay any attention to them [39, 40].

It can be seen that there is at present no system that helps users to take a decision aimed at more privacy-respecting and secure Android applications. Users must either rely on the community with comments and ratings, which rarely refer to possible security problems, or manually verify permissions and rely on their personal knowledge and understanding. Authors of previous studies, such as [89], suggest an integration of new security and privacy indicators for users.

In spite of the ineffectiveness of permission list warnings, the Android permission system seems to be a valuable source of information: 80 permissions are available to third-party applications, and this number doubles for system applications; more permissions appear with each new Android version. Information about required permissions is embedded into each application and is always available.

[1]The antivirus system proposed by Google

Instead of asking users to verify permission lists manually, we believe that an automatic analysis can be used to detect expected permissions and anomalies. With less repetitive warnings and easier indicators, users would be able to use permissions as a decision factor.

This chapter introduces the research on characterizing the behaviour of Android applications by permission requests and on permission-based risk warning system.

By this study, we test and verify multiple hypothesis:

- Application category contains similar applications that would use similar permissions. Therefore, an average or 'normal' category application could be represented by a permission pattern.
- Different application categories contain different applications. Therefore, the patterns that characterise one category should differ from the pattern characterising another category. In this case, patterns should permit to identify the category of an application by permissions this application requires.
- A pattern characterising 'normal' applications of a category should permit to measure the risk level of an application and to detect abnormal applications: applications abusively requiring permissions, bad-quality applications, applications from wrong categories and malware. By hypothesis, the more applications request permissions that are not normally observed in the category, the higher is its risk score.

The following research questions emerge from the highlighted hypothesis and are answered by this work:

- Do Android applications of different categories require different permission patterns and can be distinguished by patterns?
- Can a category pattern allow to measure an application risk/privacy level and permit malware[2] detection?

The final objective of the study is to propose a metric that evaluates the risk of a given application within a defined category. Our contribution is threefold:

First, we analyse a large set of applications collected from GooglePlay. For each category, we build a graph of 'normal' permission requests and compute graph metrics obtaining behavioural patterns.

Second, we verify which of the obtained patterns characterise categories best. We build pattern-related features and apply machine learning algorithms to classify applications into categories. The patterns containing betweenness centrality and weighted degree metrics showed better classification performances than others. Therefore, we conclude that those patterns are the most descriptive and representative for categories.

Finally, we propose to measure a privacy level of an application regarding its category and the previously obtained patterns. We suppose that normal applications will request permissions following the pattern, and applications that deviate from the pattern would more likely be abnormal: wrongly categorised, abusive or even malicious. We propose to warn users of those abnormal requests and define a threshold permitting to separate normal and abnormal applications. We evaluate the proposed warning system by injecting malicious applications into the initial dataset and verifying the performance of the

[2] An application which is specifically designed to cause harm to a system or the user, gather sensitive information, mislead user to obtain unauthorized access or to do other unwanted actions

method in malware detection. We compare the performance of our proposition with the most recent and relevant works on Android application risk evaluation and highlight the obtained gain.

The reminder of the chapter is organized as follows: Section II reminds state-of-the-art and highlight the limits of related works; Subsection III presents the methodology for category permission patterns construction, application classification and privacy score; Subsection IV presents the results of the proposed methodology; Subsection V discusses the results and considers future works. The chapter ends with a conclusion.

6.2 Related works and limits

Some works have analysed Android permissions, but with a number of limitations we aim to respond to here. Authors often use a very limited dataset for their analysis, while we perform a larger analysis using 9,512 applications from Android Market.

Most authors such as [53] identify dangerous permissions manually, while our methodology proposes an automatic pattern definition. Manual definition of dangerous permissions and their combinations can be subjective, as some permissions can be useful and malicious in different usage contexts. Some automatically identified patterns are based on the occurrences of permissions that reflect the most and the least popular permissions overall and do not describe any particular category: permissions such as Internet are highly represented in all categories [49, 55]. Rather than using the most frequent or the most rare permissions, the proposed methodology leverages the most representative and essential permissions for a particular category, maintaining the notion of functionality. We assume that applications in the same category have similar functionalities and would require similar groups of permissions, as was observed by the state-of-the-art works [55, 51]. The proposed methodology aims to highlight expected permission requests for a given category and to penalize broadly and frequently used permissions.

Many authors focus on malware detection and malware-specific patterns [57, 56, 59]. In comparison to those works, our methodology results patterns for regular applications and categories to measure the risk of applications available on the market that are not explicitly malicious but abnormal and probably abusive. Malware-based patterns uncover only currently known malware, but malware evolves over time. Comparing all applications to the expected behaviour could enable the detection of unknown malware and new abusive applications. As the methodology is based on expected behaviour, deviant requests would lead to a warning regardless of whether the given application is a malware, of poor quality, abusively requiring permissions or simply classified in the wrong category.

The proposed patterns are based uniquely on the permissions that are available on Google Play without any code extracted features that require additional computational costs. The methodology is also not limited by the maximum number of permissions that can be included in a pattern. The patterns of this work are graph-based and no previous studies on Android permissions analysis, risk warning and malware detection have used graph metrics. Finally, this study does not only evaluate the patterns in application classification but also propose and evaluate a pattern-based privacy-friendly risk warning system.

6.3 Research methodology

This section presents the research methodology and the dataset used in the study for methodology evaluation. This methodology involves three main steps:

1) the construction of patterns of expected permissions for a given category
2) the application classification to highlight the relevance of the proposed patterns
3) the construction of risk metrics that aim to detect abnormal applications

The overall objective is to provide a warning system that remains within the proposed patterns and risk metrics. The methodology is illustrated in the Figure 6.3.1.

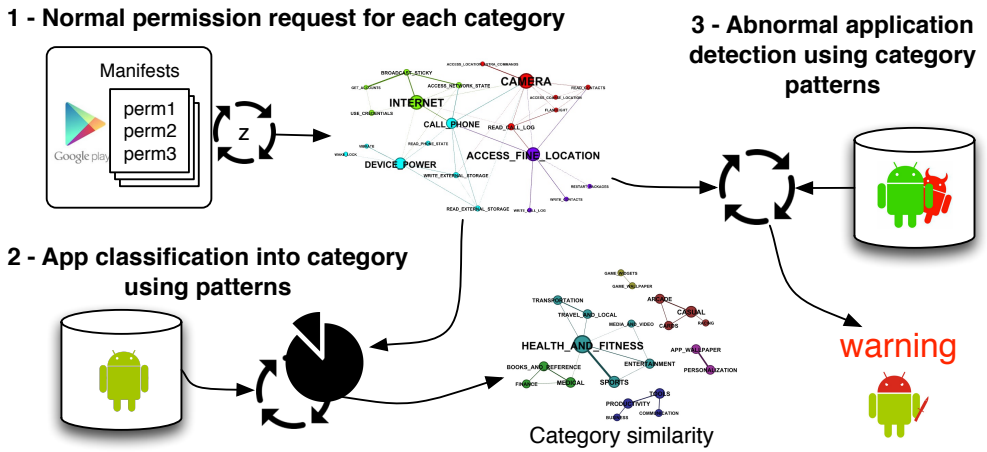

Figure 6.3.1: The summary of 3-step methodology

6.3.1 Dataset

We collected application data from Google Play market store in 2013 using a publicly available nonofficial API and a PHP script published under GNU General Public License [119]. We modified this script to make it fit our goals and stored the harvested data within a MySQL relational database. We note that this script became unusable the short time after our collection was made due to important changes made on the Google Play website.

Our database of applications contains name, description, package name, version, users' note, number of downloads, price, category, number of screenshots, author and the list of permissions as defined in the manifest. For each category, we obtained the category's name and related description.

Our crawler was able to obtain a sample of 9,512 applications related to 35 categories that contain between 190 and 590 applications each. In our sample, we observed a set of 2,133 unique permissions with 292 permissions identified as Android native permissions (263 matched the prefix "android.permissions.*" and 29 matched the prefix "com.android.*"). The other permissions are supposed to be custom.

We compared the obtained list with a list of permissions extracted from Android 4.4 and found 157 permissions that do not match currently available Android permissions. Those permissions were mainly the third party application permissions such as Mobile Device Management permissions (e.g., android.permission.sec.*), old permissions of previous Android versions (e.g., READ_SETTINGS), 2 permissions for Android in-app payment and license libraries and many wrongly written permissions.

To carry out further analysis, the dataset was filtered to omit custom, old and misspelled permissions and only kept track of permissions embedded in the Android 4.4.2 operating system - the most up-to-date Android system at the time of the study.

6.3.2 Permission usage pattern construction

We assume that applications grouped into categories provide similar functionalities and would therefore require similar sets of permissions. This section presents the methodology for building the permission patterns relevant to each category. Instead of analysing individual permissions, the methodology investigates co-occurring pairs of permissions, building permission graphs. The application of graph analysis metrics is, therefore, possible to build large patterns without limiting the number of nodes involved. An overview of the pattern identification methodology is presented in Figure 6.3.2.

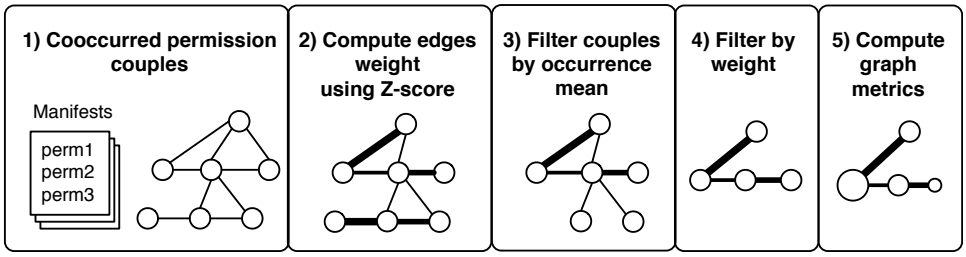

Figure 6.3.2: Five steps of pattern construction methodology.

First step - build permission pairs: if two permissions are seen together in the Android Manifest of one application for a particular category, they are linked together (step 1 in Figure 6.3.2).

For each category Cat, a graph denoted $G_{Cat}(N_{Cat}, E_{Cat})$ is created where the set of nodes N_{Cat} represents the permissions, and the set of edges E_{Cat} represents two commonly used permissions in the category.

To avoid over-represented couples, such as ACCESS_NETWORK_STATE and INTERNET, the importance of a permission pair (i.e. edge) in the category is calculated regarding its overall usage: mean and standard deviation. For this purpose, the second step compute the $\mathcal{Z} - score$ (a.k.a. Standard score) for each pair of permissions observed for each category. This score highlights not only the most commonly used permissions in the category, but also the most representative permissions, thereby characterizing each category. The calculated $\mathcal{Z} - score$ represents the edges weight in the graph (step 2 in Figure 6.3.2) and is defined by equation 6.3.1.

$$\mathcal{Z}_{pp}^{\mathcal{C}} = \frac{\mathcal{F}_{pp}^{\mathcal{C}} - \mu_{pp}}{\sigma_{pp}} \tag{6.3.1}$$

Where :

pp is a permission pair $pp \epsilon N_{Cat}^2$.

$\mathcal{F}_{pp}^{\mathcal{C}}$ is the frequency of the pair of permissions pp in the category Cat: occurrence of the pair pp weighted by the total occurrence of all pairs in the category.

μ_{pp} and σ_{pp} are the mean and standard deviation of the pair pp across all categories.

Manual investigation of the dataset revealed the presence of very poor and possibly malicious applications, which would be the reason for unexpected permissions obtaining a high $\mathcal{Z} - score$. Google Play is known to host those types of applications, and as

such their presence in our dataset is not surprising. To avoid abnormal applications and insignificant permissions during the pattern construction step, the third step omit permission pairs with a mean occurrence below one (step 3 in the figure 6.3.2).

To extract patterns, the final graphs are filtered by edge weights, keeping only the most category-relevant permissions (step 4 in the figure 6.3.2).

By definition, the $\mathcal{Z} - score$ will be negative if the observed frequency of a permission pair in a category is below the mean. The $\mathcal{Z} - score$ will be equal to one if the observed frequency is higher than the mean by exactly the standard deviation measure. Finally, the $\mathcal{Z} - score$ will be higher than 1 if the frequency of a given permission pair is significantly higher than the mean. The fourth step filter the category graph to only keep edges weighted above 1; non-connected nodes are also omitted. The graph obtained for a given category is referred to in this chapter as a pattern.

A pattern for a category Cat is denoted as:

$$Pattern_{Cat} = G_{Cat}(N_{Cat}, E_{Cat}) \tag{6.3.2}$$

For each node belonging to N_{Cat}, the fifth step computes the following graph analysis metrics: node degree (denoted D), weighted degree (denoted WD), betweenness centrality (denoted B), closeness centrality (denoted C), PageRank score (denoted PR), Hub (denoted Hub) and Authority (denoted $Auth$) scores [120] (step 5 in the figure 6.3.2). The underlying assumption is that these metrics could provide more measurable values of the importance of a given permission for a given category. These metrics provide different ways to express the 'importance' of a permission in a pattern, thus providing a potential insight into what type of metric must be considered when addressing our research issue.

The **degree** of a node is the number of edges incident to the node. In the case of Android permissions, the degree represents the number of relative pairs that the permission is a part of. The underlying assumption is that the permissions often used in conjunction with many other permissions are the most relevant.

The **weighted degree** of a node is a degree of a node weighted by the strength of each edge. In this work, the strength is represented by the $\mathcal{Z} - score$ and the weighted degree will show if the node is met in high or low scored relative pairs and/or in few or many different pairs. The underlying assumption is that permissions significantly used with many other permissions are the most relevant to our analysis.

The **betweenness centrality** of a node represents the number of shortest paths between all nodes that go through the given node. The relevant permissions are, in this case, the permissions that often appear on the shortest path between any permissions. These permissions may be viewed as a bridge between different communities. By assuming that communities can be related to the functionalities of applications, the important permissions are those that are required in many different functionalities of an application.

The **closeness centrality** represents how far a node is from all the other nodes. The closeness centrality is calculated as a reciprocal of the sum of distances between a given node and all other nodes. Under this assumption, the important permissions are often used in combination with other permissions, and if not, they are only separated from other permissions by a few intermediaries.

The **PageRank** score is a recursive algorithm in which a high score of a node depends on the incoming links from nodes that are themselves important [121]. Important permissions are often used in common with important permissions. The assumption is that permissions with a high PageRank cannot be circumvented.

Node measure	**Formula** $m(p)$
Degree	$D(i) = \sum_{j=1}^{\lvert N_{Cat}\rvert} a_{i,j}$ Where : $a_{i,j}$ is the element located at the i^{th} row and j^{th} column of the binary adjacency matrix.
Weighted degree	$WD(i) = \sum_{j=1}^{\lvert N_{Cat}\rvert} w_{i,j}$ Where : $w_{i,j}$ is the weight between node i and node j.
Betweenness	$B(i) = \sum_{s \neq i \neq t} \frac{\sigma_{perm_s,perm_t}(i)}{\sigma_{perm_s,perm_t}}$ With : $\sigma_{s,t}(i)$ is the number of shortest paths between s, and t passing through i. and $\sigma_{s,t}$ is the number of shortest paths between s and t.
Closeness	$C(i) = \frac{1}{\sum_{j=1}^{\lvert N_{Cat}\rvert} d(j,i)}$ With $d(j,i)$ the distance between node i and node j.
PageRank	$PR(i) = \alpha \sum_j a_{ji} \frac{x_j}{L(j)} + \frac{1-\alpha}{N}$ With $L(j) = \sum_j a_{ij}$ and α a predefined residual probability.
Hub-Authority	$\begin{cases} \text{Hub}(i) = \sum_j \text{Auth}(j) \\ \text{Auth}(i) = \sum_j \text{Hub}(j) \end{cases}$ With the following initial conditions : $\forall$i, Auth$(i) = Hub(i) = 1$

Tableau 6.1: Set of node measures tested in this study

The **Hub** and **Authority** scores are the outputs of the Hyperlink-Induced Topic Search (HITS) algorithm where a good hub has many outgoing edges to authorities and a good authority node has many incoming edges from hubs [122].

The set of metrics that can be applied to a node belonging to N_{Cat} for a category Cat is denoted as follows:

$$\mathcal{M} = \{D, WD, B, C, PR, Hub, Auth\} \tag{6.3.3}$$

Each metric $m \epsilon \mathcal{M}$ for a permission $p \epsilon N_{Cat}$ is calculated using the formula $m(p)$ as summarized in Table 6.1.

The pattern construction methodology produces lists of relevant permissions for each category, in which 7 graph analysis metrics are associated with each permission in each list. The metrics $m(p)$ are used as inputs for classification feature construction.

6.3.3 Classification of applications into categories

In order to ensure that the patterns represent a particular category, it was decided to measure their efficiency in application categorization well. Multiple hypotheses related to the pattern metrics were tested in order to identify the most relevant graph metric. The performances are also compared with state-of-the-art work.

6.3.3.1 The application classification problem

The problem of application categorization can be formalized as follows: given an application with an unknown category, can the system predict the actual category of this application? Although the actual class of an application is known, in order to confirm the efficiency of the proposed patterns, it is important to identify whether the actual class corresponds to the predicted class.

Basic metrics for characterizing the quality of the prediction rely on true positives, false positives, false negatives and true negatives. These values are often modeled in the form of a confusion matrix, as shown in table 6.2.

Many performance metrics may be calculated from such confusion matrices, such as the true positive and false positive rates. The True Positive Rate (TPR) is defined as the number of correctly classified positive instances (TP) divided by the sum of the true positives (TP) and false negatives (FN). For example, in terms of classifying applications in the Finance category, this score equals the number of correctly identified finance applications divided by the total number of finance applications ($TPR = TP/(TP + FN)$). The False Positive Rate (FPR) is defined as the number of false positive instances (FP) divided by the number of False Positives (FP) and True Negatives (TN). For the finance category, this score equals the number of applications correctly classified as not belonging to the finance category divided by the total number of applications known as not belonging to the finance category ($FPR = FP/(FP + TN)$).

		Predicted class	
		Positive	Negative
Actual class	Positive	True Positives (TP)	False Positives (FP)
	Negative	False negatives (FN)	True Negatives (TN)

Tableau 6.2: Confusion matrix for category classification

The Receiver Operating Characteristics (ROC) curve captures the performances of a classification by analysing the relationship between the False Positive and the True Positive rates. This curve is obtained by plotting each point (TPR, FPR) for multiple threshold value, thereby highlighting the global performance of the classifier. The Area Under the ROC curve (AUC) is often used for performance comparison, and shows how good a classifier is compared to a perfect classifier (AUC score of 1) and random classifier (AUC score of 0.5) [123].

The F-measure (a.k.a F_1 score) is another performance value of a classifier that is calculated as the harmonic mean of precision ($TP/(TP + FP)$) and recall ($TP/(TP + FN)$). The more general F_x measure allows to set up the relative importance of precision with regard to recall.

The performances of the proposed methodology were tested using a 10-fold cross-validation. The dataset was randomly divided into 10 equal subsets. The training was performed on 9 folds and the actual classification was performed on the last fold. This process is repeated for several rounds, and helps to avoid data overfitting problems.

The dataset contained different numbers of applications for different categories and a misbalance in the number of class members can bias the classification. To avoid this bias, the data was pre-processed with a distribution-based balancer [124]. This procedure resulted with 300 members for each category.

Several classification methods were tested such as Random Tree, Support Vector Machine, Naive Bayesian, etc. Only the Naive Bayesian method is presented for clarity in this chapter, as this method provided the best classification results.

6.3.3.2 Features selection

Classification features are typically the recurrent data found for all data entries, regardless of their class, such as profile data for a person's classification, word occurrence for document classification, or a presence of permission for Android application classification. In this study, each class has a corresponding pattern and the data should be classified into all categories according to those patterns, therefore a different approach for feature construction is required to perform such a classification.

Similarity features denoted Sim are built for each application by comparing the list of permissions required by an application with the patterns of each category.

A_{perms} is a set of permissions required by an application A and N_{Cat} is a set of permissions belonging to a pattern in the category Cat. The first feature is a number of common permissions between a given application and a category pattern defined as $PermCount$, and its score is defined in equation 6.3.4.

$$Sim_{PermCount}(A, Pattern_{Cat}) = |A_{perms} \cap N_{Cat}| \tag{6.3.4}$$

Other features are built using the graph metrics presented in Table 6.1 and are measured as a sum of scores for each metric for all common permissions between an application and a pattern.

A similarity feature for a metric m corresponds to the normalized similarity between an application permissions list and a pattern in category Cat as follows:

$$Sim_m(A, Pattern_{Cat}) = \frac{\sum\limits_{p \epsilon \{A_{perms} \cap N_{Cat}\}} m(p)}{\sum\limits_{p \epsilon N_{Cat}} m(p)} \tag{6.3.5}$$

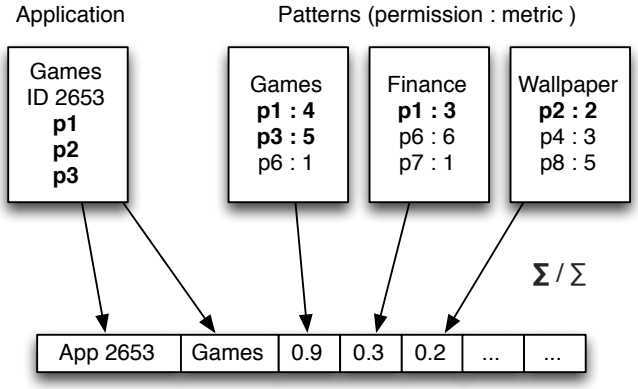

Figure 6.3.3: Example of feature construction for one application, one metric and patterns.

where $m \epsilon \mathcal{M}$

For each application, a similarity feature for each category and each graph metric is built. An example of a feature set for a metric is shown in Figure 6.3.3. The application 2653 originally from the 'Games' category requires three permissions: p1, p2 and p3. This permissions list is compared with the permissions for each category pattern; the patterns for 'Games', 'Finance' and 'Wallpaper' categories are represented in the figure. The 'Game' pattern contains three permissions: p1, p3 and p6, where two permissions p1 and p3 are common to the application. The similarity feature for the 'Games' category would be equal to the sum of scores for common permissions p1 and p3 $(4 + 5)$ weighted by the total pattern score $(4 + 5 + 1)$. The 'Finance' pattern contains only one common permission p1, bringing the similarity to 3 out of 10. Likewise, the similarity score for the 'Wallpaper' category is 2 (p2) out of a total 10.

Finally, one application and one metric obtain as many features as patterns (categories).

6.3.4 Privacy score and risk metrics

This section explains the methodology for building a privacy score using the best identified features from the feature verification step, and discusses the risk warning.

A privacy score is based on the following assumptions:

1. The more an application is similar to a pattern, the higher the privacy score.
2. The score of an application must be high if an application requires central permissions in the corresponding pattern.
3. If an application requires permissions that are not represented in the pattern, its score must decrease.

Following those assumptions, the similarity between an application and a category should be weighted by the number of permissions required by an application. As two features were identified as the most valuable, a similarity between an application and a pattern is defined as the likeliness LN combining the similarity of both features as shown in equation 6.3.6.

$$LN(A, Cat) = Sim_{m1}(A, Pattern_{Cat}) + Sim_{m2}(A, Pattern_{Cat}) \tag{6.3.6}$$

where $m1 \epsilon \mathcal{M}$, $m2 \epsilon \mathcal{M}$ (from the equation 6.3.3).

The privacy score, denoted $Privacy$, of an application A in a given category Cat is defined in equation 6.3.7.

$$Privacy(A, Cat) = \frac{LN(A, Cat) - LN_0}{|A_{perms}|^{\beta}} \tag{6.3.7}$$

Where $\beta \epsilon \mathbb{R}^+$ is a parameter permitting adjustment of the risk threshold and LN_0 is a minimal accepted likeness.

An ideal application will perfectly follow the corresponding category pattern: if such an application requires only two permissions, it will use the top 2 permissions in the pattern (the permissions with the highest 'importance'); if this application requires 5 permissions, it will use the top 5 permissions in the pattern, etc. Even if real applications do not ideally follow the pattern because of widely used permissions, some applications would be closer to the pattern than others, and would be preferable for the user.

Figure 6.3.4 represents the risk thresholds for different β. The x axis represents the number of permissions $|A_{perms}|$, the y axis represents the pattern likeness score $LN(A, Cat)$, the red line represents examples of thresholds using different β, the red dotted line represents the risk threshold with a defined minimal likeness LN_0, the bold black line represents an ideal similarity score for an application following the pattern perfectly, the grey dots are applications in the dataset and the red dots are the applications producing a warning.

When β is set to 0, the privacy score would be equal to the likeliness score and the risk threshold would be fixed at a given likeliness. Setting β to 1 defines a linear separation: the threshold would be determined by a given average likeliness score provided by permissions in the application. When β is higher than 1, the risk threshold can be illustrated as in figure 6.3.4. This threshold permits a risk to arise only for the least similar applications, and more an applications have permissions, higher the similarity threshold.

An application having a $Privacy$ score below a defined threshold would lead to a risk warning.

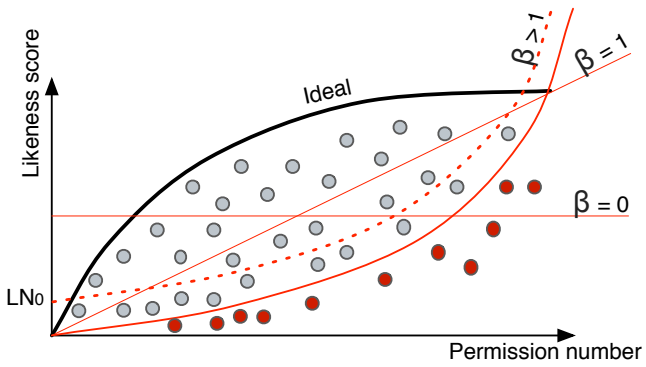

Figure 6.3.4: Risk threshold representation for different settings of α; applications colored in red would raise the warning for $\alpha > 1$

6.4 Results

This section presents the result for the pattern construction, evaluation and application methodology. First, the patterns obtained for each category are discussed and a sample of 6 category-patterns are detailed. The graph-related metrics are then compared to a

simple occurrence regarding the category representation: the finance category is used for comparison. Then, the performance of metrics and classification results are presented in subsection 4. Subsection 5 presents the result of the abnormal application detection using the defined *Privacy* score and different thresholds. Finally, the secondary results on category similarity obtained using the classification confusion matrix are presented.

6.4.1 The category patterns obtained

This section presents the results for the patterns obtained and discusses some central permissions for a sample of categories.

Table 6.3 illustrates the number of permissions and permission pairs left for each pattern after the execution of the methodology presented in Figure 6.3.2. First of all, two categories were dropped: 'brain' and 'education' which contained 228 and 244 applications respectively. The applications in these categories required some very rare permissions and/or non-relevant pairs. In the absence of the pattern, we assume that the normal applications for the 'brain' and 'education' categories would not require any permissions. We also note the possibility that at the time of data collection, those categories did not contain high quality applications related to the categories. This hypothesis is supported by the fact that the 'brain' category no longer exists on Google Play, and more fine-grained categories such as puzzle_games and educational_games appeared instead. In the dataset, educational games are found in both the 'brain' and 'education' categories, with puzzles and language learning applications.

Category	**Nodes**	**Edges**	**Category**	**Nodes**	**Edges**
communication	59	835	weather	23	53
app_widgets	61	602	casual	17	52
productivity	61	475	photography	22	40
tools	58	436	finance	22	40
social	44	260	medical	26	40
personalization	37	221	media_and_video	22	39
app_wallpaper	29	183	health_and_fitness	21	35
entertainment	30	143	lifestyle	28	32
travel_and_local	37	140	game-wallpaper	19	29
business	49	138	transportation	19	24
music_and_audio	29	97	books_and_references	19	22
libraries_and_demos	21	79	cards	13	12
shopping	22	56	sports	12	9
comics	24	55	news_and_magazines	9	7
arcade	19	53	sports_game	7	7
game_widgets	20	53	racing	5	3

Tableau 6.3: Number of permissions and co-required permissions by pattern.

Some patterns, such as 'communication' and 'app_widgets' contain a large number of nodes (permissions) and edges (relative pairs). This large number shows either that applications in these categories are highly multifunctional and often require many permissions, or that the category contains many applications with different functionalities.

This might be an indication that these categories are too broad, and could be divided into subcategories.

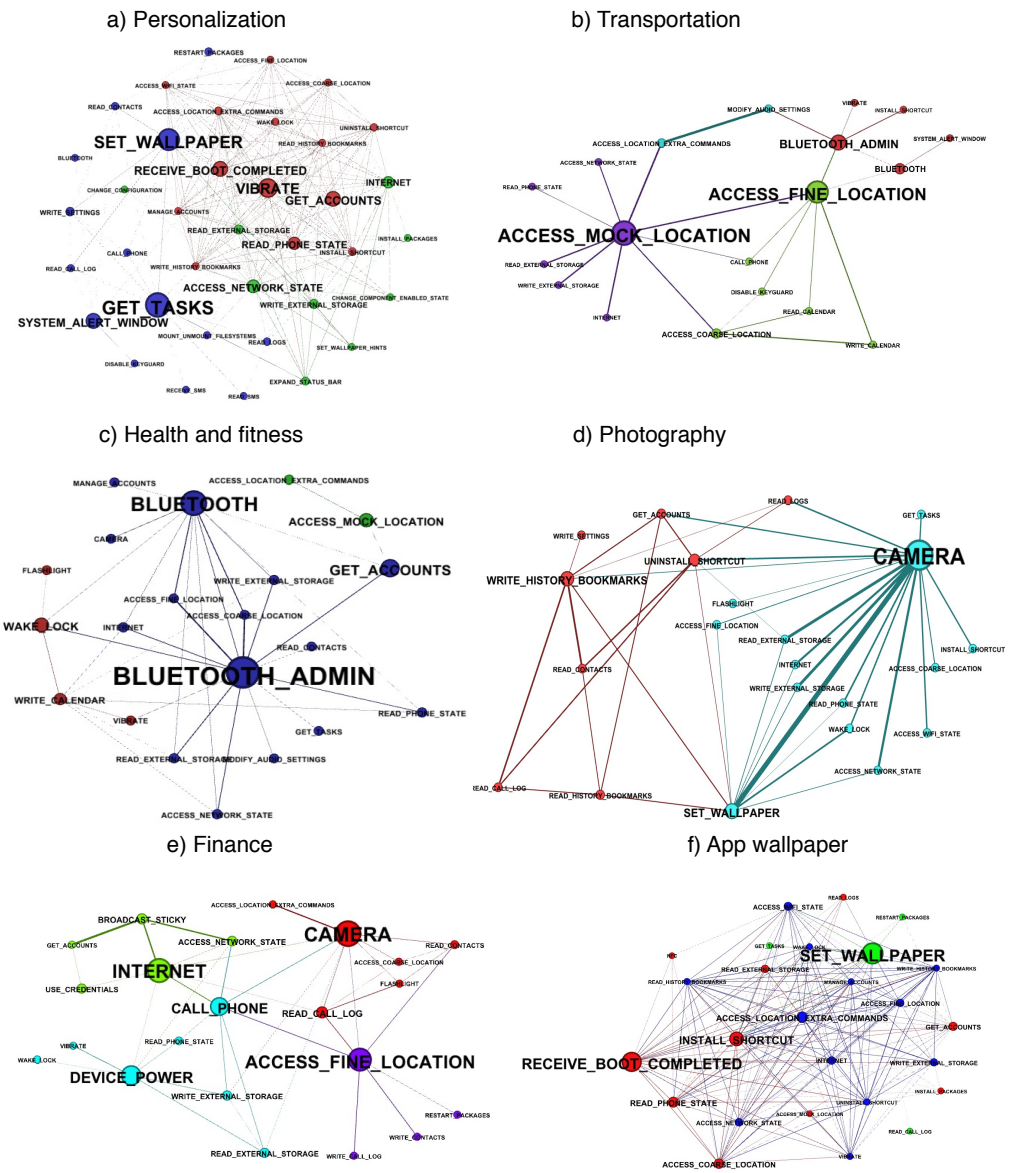

Figure 6.4.1: Examples of obtained permission patterns for fours categories. Colors represent the modularity classes, node size represents betweenness centrality score and edge thickness represents the weight.

Figure 6.4.1 illustrates the patterns obtained for six different categories: personalization, photography, transportation, health and fitness, finance and wallpaper. Each pattern has been visualized by the Gephi graph visualization software [125]. For each pattern, the colour of the nodes is defined by the modularity class of the node: permissions belonging to the same modularity class have the same color. The size of the nodes is defined by their betweenness centrality: further analysis shows the betweenness centrality to be the most relevant metric for our methodology. The labels of nodes correspond to Android permission names, and are proportional to the nodes' size. We used the Fruchterman Reingold [126] and the Force atlas 2 [127] algorithms for graph visualization, as well as manual adjustments improving the labels' visibility.

The first pattern is related to the 'Personalization' category (denoted a) in Figure 6.4.1. 'Personalization' applications mostly involve screen personalization by wallpapers that is expressed by the most central permission - set_wallpaper. Another central permission is get_tasks that enables recently run applications to be obtained and proposes in-context personalization. System_alert_window and vibrate permissions offer personalized notifications.

The 'Transportation' category is denoted b) in Figure 6.4.1. Access_fine_location is clearly the most important permission for transportation, as most applications in this category are maps and GPS-related software. Access_mock_ location is a development permission used for testing. It seems that most of applications in our dataset have kept this permission in production and some applications offer a fake GPS position within an application using this permission. Bluetooth_admins is one of the central permissions in the 'Transportation' category: GPS and car-related tools often use this permission to connect with a hands free Bluetooth kit or specific car diagnostic devices.

Figure 6.4.1 c) shows the pattern obtained for the 'Health and fitness' category. The most central permissions - bluetooth and bluetooth_admin - clearly indicate the applications in this category are mostly used with health and sport-related devices such as an accelerometer, heart and step monitor, etc.

The 'Photography' pattern is shown as d) in Figure 6.4.1. The most central 'Photography' permission is camera for accessing and using a device-embedded camera to take photographs. 'Photography' applications use set_wallpaper permission with a camera enabling the picture taken to be used as a wallpaper.

The wallpaper category ('app_wallpaper') is shown as f) in Figure 6.4.1 and gives a high number of significant permissions due to the diversity of animated wallpapers and functionalities accessed and provided by animated wallpapers. 'Set_wallpaper' is the central permission; the other wallpaper-related permissions are also in the pattern. Filesystem and package management permissions are related to different personalizations proposed by single wallpaper application as well as shortcut and widget management permissions. Many functionality-related permissions are seen permitting fast access from the wallpaper to build in functionalities: phone calls, SMS, calendar, settings, application list, contacts, bookmarks, cache. External storage permissions allow storing personalization images locally and network related permissions allows getting additional information, such as weather, and download new images.

The 'Finance' category is represented as e) in Figure 6.4.1. The most central permissions for finance category are 'call_phone' and 'internet'. Permissions used for calls including voip calls and SMS are often available for contacting the bank or a service manager. 'Internet' permission seem necessary to access to up-to-date banking information. One can distinguish many account and authentication linked permissions due to the sensitivity of the financial information and the necessity for a secure usage. Localization permissions also appear in the pattern, probably to apply different location-dependent payment criteria or to identify the nearest offline office. Camera is often used for QR-codes and cheque deposit in finance applications.

The interactive graphs for all categories can be found in [128].

6.4.2 Graph centrality features vs. occurrence

The most commonly used permissions indicator from the state-of-the-art is the simple occurrence of permissions in applications in different categories. To highlight the im-

provement of the proposed methodology with respect to these works, the results for the 'Finance' patterns are compared to the top 5 most frequent permissions obtained for the same category.

Table 6.4 shows the top 10 permissions and the percentage of applications of the dataset that require these permissions. INTERNET permission is the most frequently required one as observed by previous works.

Permission	**Applications %**
INTERNET	91,88%
ACCESS_NETWORK_STATE	83,31%
WRITE_EXTERNAL_STORAGE	60,39%
READ_EXTERNAL_STORAGE	60,29%
READ_PHONE_STATE	49,92%
WAKE_LOCK	33,01%
ACCESS_WIFI_STATE	31,47%
VIBRATE	30,07%
ACCESS_COARSE_LOCATION	27,73%
ACCESS_FINE_LOCATION	27,42%

Tableau 6.4: Top 10 of permissions usage. Each permission is originally prefixed by 'android.permission".

Permission	**Occurrence %**	**Betweenness**
INTERNET	91,19 (Rank 1)	361 (Rank 2)
ACCESS_NETWORK _STATE	75,15 (Rank 2)	202 (Rank 6)
WRITE_EXTERNAL _STORAGE	49,32 (Rank 3)	98 (Rank 16)
READ_EXTERNAL _STORAGE	49,12 (Rank 4)	98 (Rank 17)
READ_PHONE _STATE	32,88 (Rank 5)	69 (Rank 19)

Tableau 6.5: Top 5 frequent permissions for the 'Finance' category

Permission	**Occurrence %**	**Betweenness**
CALL_PHONE	11,74 (Rank 12)	470 (Rank 1)
INTERNET	91,19 (Rank 1)	361 (Rank 2)
CAMERA	10,96 (Rank 14)	360 (Rank 3)
USE_CREDENTIALS	3,52 (Rank 21)	257 (Rank 4)
ACCESS_COARSE _LOCATION	19,18 (Rank 8)	231 (Rank 5)

Tableau 6.6: Top 5 permissions according to betweenness centrality for the 'Finance' category

Tables 6.5 and 6.6 show the permissions of the 'Finance' category with the corresponding occurrence and betweenness centrality scores. One can see that top 5 Finance permissions from the Tables 6.5 are equal to top 5 permissions for all categories presented in the Table 6.4. This shows that even if those permissions are highly used in the category they are not representative of it. The proposed pattern contains those permissions but not as

highly ranked. The top 5 permissions defining 'Finance' as shown in the proposed results in Table 6.6 mostly propose online (internet) services and needs secure authentication (use credentials). Banking applications prefer including direct contacts with the bank in the application (call phone), allows check deposit (camera) and propose office or cash withdrawal locations (access coarse location). The pattern is more accurate than simple frequency analysis to define a particular category. The use of the $Z-score$ is particularly adapted for this purpose since it allows to measure how relevant a couple of permissions is to a category with respect to its overall permissions use in units of standard deviation.

6.4.3 Classification and features accuracy

This section provides the results of the Naive Bayesian algorithm for classifying the applications into categories using pattern-related features. The suitability and efficiency of these features are analysed separately and in combination. The obtained results are also compared with the state-of-the-art method. The pre-processing and classification is performed with WEKA [129].

The first step was to perform a classification using the state-of-the-art method. Our patterns are built based on the permission lists, it is important to compare the performance of those patterns in describing a category comparing to the initial permission dataset as we can suppose the initial permission lists already contain all the information and category description. Moreover, previous state-of-the-art works often include simple permission list while classifying Android applications [130, 131, 56]. A permission binary vector of 229 Android 4.4 permissions in built for each application, in which a value of 1 is attributed to permissions required by a given application and 0 otherwise. The Naive Bayesian algorithm showed relatively poor results with only 15% of instances correctly classified. The results are shown in the last line in Table 6.7 (average scores for all categories).

The second step was to classify separately the applications using the proposed pattern-related features for each metric. The middle block of lines in Table 6.7 shows the results of this step. The betweenness centrality shows the best performance with an F-measure equal to 0.417 and AUC of 0.941. This result confirms the hypothesis that permissions with high betweenness centrality are necessary for the main functionality or common to many functionalities of the category, and permissions with high betweenness centrality represents the category well. Betweenness centrality outperforms weighted degree by 0.086 and 0.023 in the F-measure and AUC respectively. The PermCount feature, which only uses patterns without integrating graph metrics, does not perform as well but surpasses binary vectors by 0.118 in the F-measure and 0.171 in the AUC.

The third step combined features from the most to the least efficient to measure the gain obtained by each metric. Table 6.7 combines all the classification results ordered by true positive rate. The top six lines represent the combined features. Figure 6.4.2 shows the gain in F-measure and AUC brought by each metric added one by one, starting from the betweenness centrality. The highest gain in the AUC is brought by combining the two best metrics: betweenness centrality and the weighted degree. The gain increases with each additional metric, but the gain for the next 5 metrics is smaller than the gain brought by the weighted degree alone. The most significant gain regarding the F-measure is provided by the weighted degree (+0.148).

Figure 6.4.3 shows the F-measure results for each classification category and different metrics. The results for the state-of-the-art method are denoted 'Binary' (empty cross).

Metric	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-measure**	**ROC Area**	**Correctly classified (%)**
All 7 metrics	0.809	0.006	0.818	0.809	0.809	0.995	80.86
Betweenness & Weighted & Degree & PageRank & HubAuth & PermCount	0.787	0.007	0.794	0.787	0.786	0.994	78.66
Betweenness & Weighted & Degree & PageRank & HubAuth	0.769	0.007	0.78	0.769	0.77	0.993	76.87
Betweenness & Weighted & Degree & PageRank	0.712	0.009	0.731	0.712	0.715	0.99	71.21
Betweenness & Weighted & Degree	0.653	0.011	0.681	0.653	0.657	0.985	65.34
Betweenness & Weighted	0.559	0.014	0.599	0.559	0.565	0.973	55.85
Betweenness	0.412	0.019	0.46	0.412	**0.417**	**0.941**	41.23
Weighted Degree	0.338	0.021	0.329	0.338	0.331	0.918	33.75
Degree	0.322	0.022	0.312	0.322	0.315	0.914	32.20
PageRank	0.315	0.022	0.306	0.315	0.308	0.911	31.54
Authority/Hub	0.308	0.022	0.299	0.308	0.301	0.908	30.81
Closeness	0.27	0.024	0.26	0.27	0.263	0.89	27.04
PermCount	0.265	0.024	0.256	0.265	**0.258**	**0.891**	26.54
Binary	0.15	0.03	0.15	0.15	**0.14**	**0.72**	15.24

Tableau 6.7: Classification results for each metric and combinations of metrics

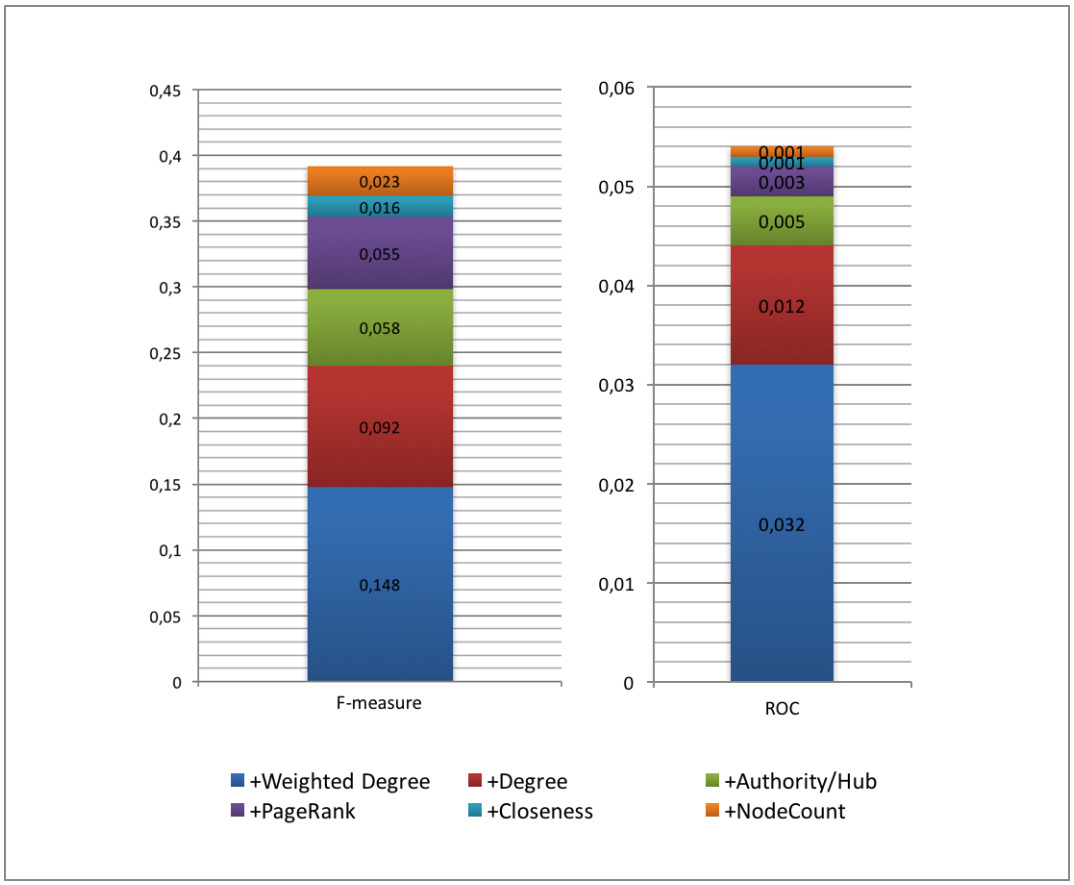

Figure 6.4.2: Performance gain brought while adding all metrics one by one to betweenness centrality

The other lines represent the results for the previously presented metrics: the red line illustrates betweenness centrality results (squares); the green line represents application permissions and pattern intersection (triangles); the blue line illustrates the combination of two best metrics (full cross), and finally, dark-blue shows the final results for all the metrics combined (diamonds).

One can see that for most categories, the F-measure for pattern extracted features is higher than the binary vector. Even the feature exploring only common permissions between an application and patterns generally obtains higher results than the binary approach, and graph analysis metrics obtain a significantly better classification performance. These observations show the relevance of patterns in category definition and the value brought by graph modeling and analysis.

Some categories, such as 'Lifestyle', 'Health and fitness' and 'Racing' have low scores in binary classification; in spite of the improvement in accuracy, the same categories remain the hardest to classify with betweenness centrality; other metrics help to significantly raise the F-measure. Categories, such as 'Game_wallpaper', 'Game_widgets', 'Finance' and 'App_widgets' - one of the hardest to classify with binary features - obtain high gains with the proposed pattern methodology. The classification for the 'News and magazines' category is particularly boosted by the weighted degree metric. A similar boost is seen for the 'Lifestyle' and 'Business' categories, while other metrics are added to betweenness centrality and the weighted degree.

Those results allow us to conclude that among all metrics, patterns containing betweenness centrality and weighted degree represent categories best. Combined together, those patterns perform well in classifying Android applications and bring the biggest performance gain comparing to other added metrics. The results for the initial permission list is low that justifies the pertinence of the methodology of pattern construction. We consider our second hypothesis as confirmed and the response to our first research question as positive.

6.4.4 Risk warning for suspicious applications

This section presents the results related to suspicious application detection and risk warning using the privacy score. The classification results show that the best features are betweenness centrality and weighted degree. The both features are used to obtain a more accurate privacy score. By including both features in equation 6.3.6, each permission of the pattern becomes considered.

In order to identify the best threshold and β values for equation 6.3.7, applications in the photography category were manually tagged: '1' is attributed to applications that seem abusive in terms of permissions or use abnormal permissions, and '0' otherwise.

Some applications that apparently should not belong to the 'photography' category were also tagged with '1' . For example, applications such as photo galleries for wallpapers clearly belong to the 'wallpaper' category and photo sharing social networks belong to the 'social' category. The application name and description from the dataset and, in some cases, the profile and screenshots from Google Play were used to identify suitable tags. To obtain as more significant and diverse dataset of risky applications, we injected a sample of 100 known malware applications into the photography dataset and tagged them as 'risky' (tag '1').

We have tested different β and thresholds to evaluate the performance of the risk warning using the tagged instances. Figure 6.4.5 depicts the best F-measure results for

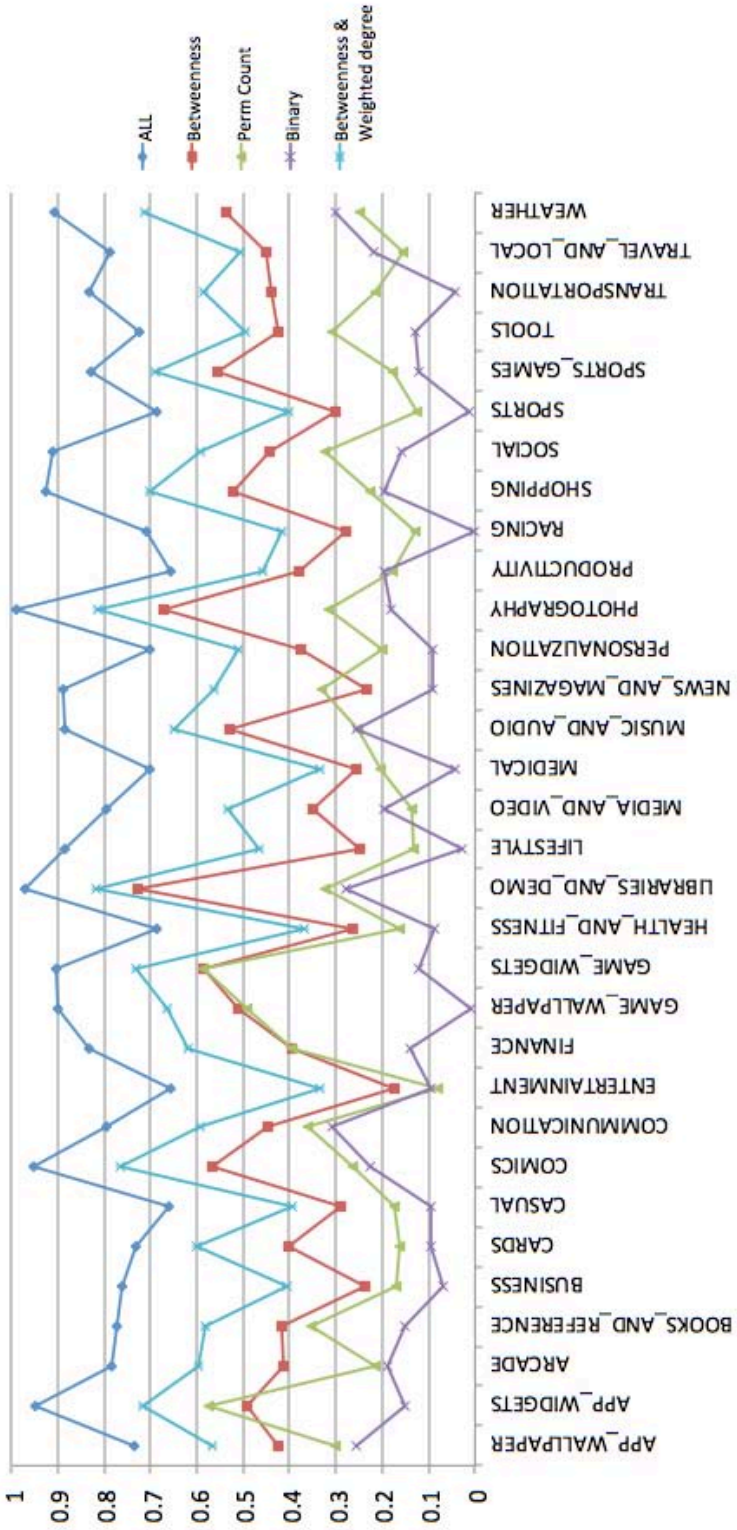

Figure 6.4.3: Binary vector and pattern-related features comparison regarding F-measure and all categories

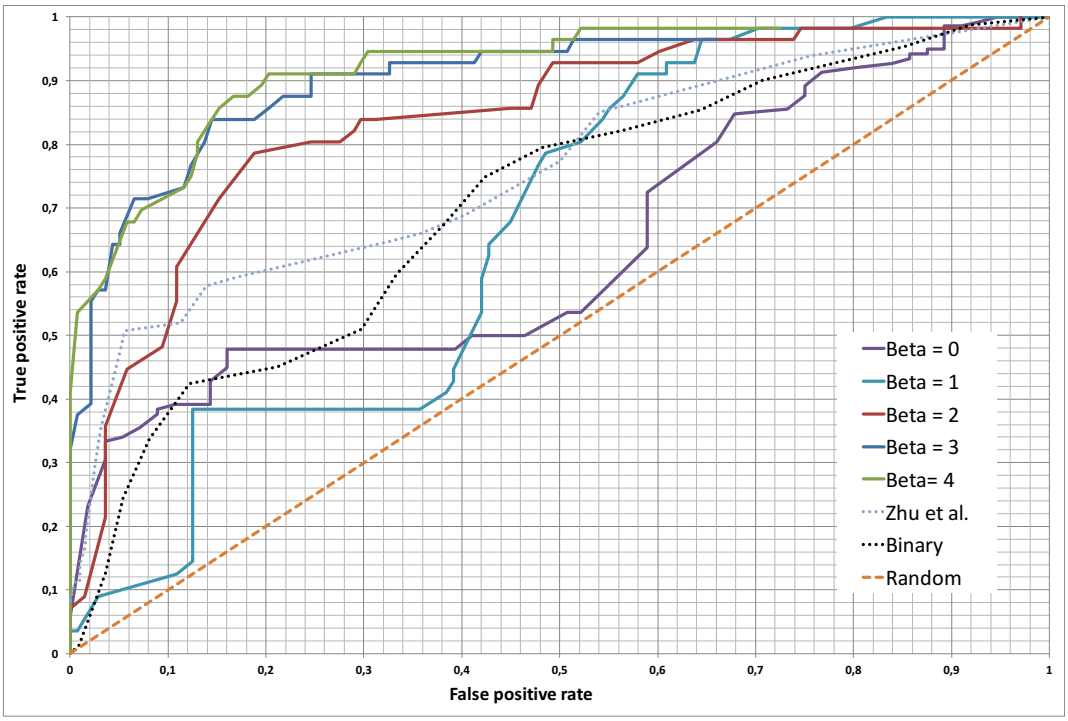

Figure 6.4.4: Performance for risky application detection in 'Photography' category. ROC curves for for different β (denoted Beta on the graph) and thresholds for the equation 6.3.7.

different values of β and minimum likeness. We also compared our results with the risk detection, using only the number of required permissions as a very high β becomes equivalent to this indicator. It is apparent that the permission number indicator shows low results, as many injected applications requested a small number of permissions. The only likeness-based threshold also shows low results that are similar to permission number-based detection. Privacy-based best results for the 'photoghrahy' category are seen for the minimal likeness of 10. The β value affects the performance, which increases for each β until it reaches its peak at $\beta = 3$, and the performance then drops again with $\beta = 4$ and $\beta = 5$. The results justify the need for an adjustment parameter and a minimum likeness parameter of the privacy score.

Figure 6.4.6 shows the optimal threshold curve. The x-axis represents the number of Android 4.4 permissions used by an application. The y-axis represents the likeness of an application calculated using equation 6.3.6. The black dots are 'non risky' applications located above the threshold; the red dots are 'risky' applications situated below the risk threshold. The labels indicate the manual tags: 'N' for normal applications and 'R' for risky applications. There is a great diversity among risky applications, while normal applications are similar in their permission requests. Normal applications often have identical likeness as they require identical permissions. The more permissions required by an application, the more severe the threshold; minimal likeness threshold enables leverage of an application that requires few non-central permissions in common with the pattern.

Table 6.8 provides detailed results not only for the F-measure, but also for the F_2-measure and the $F_{0.5}$-measure: the F_2-measure gives more value to recall that is the importance of not raising false negatives; the $F_{0.5}$-measure gives more value to precision that is the importance of not raising false positives. The best scores are shown in bold

and the smallest scores are shown in italics. The permission number gives the best F_2-measure considering all applications having more than 9 permissions to be risky. In fact, some popular applications request many permissions due to additional functionalities. Users can find such assumptions simplistic and even annoying if the false positive rate is high. Users are known not to pay attention to repetitive warnings, and so the number of false positives should be reduced.

We compare the performance of the proposed risk warning system with the state-of-the-art studies that measure the risk level of Android applications. The authors of the latest study propose an approach based on random walk and an application-permission bipartite graph [66]. The authors model Android applications as a bipartite graph where nodes representing applications are linked to nodes representing permissions that are required by each application. The edges are weighted using the equation 6.4.1 and the final risk score is obtained by applying random walk. We apply the method proposed in [66] to our dataset with the injected malware and keep 'Photography' category for performance comparison. The obtained performance is plotted in Figure 6.4.4 as a light-blue dotted line.

$$w_{ij} = \frac{f_{ij}}{\sum\limits_{e_{ik} \in E} f_{ik}} \tag{6.4.1}$$

where f_{ij} is the number of Apps in category $c(a_i \in c)$ requesting permission p_j.

Many studies use a selection of dangerous permissions or permission lists for risk and malware detection [55, 54, 132, 131, 59]. Therefore, we also verify the performance of detecting abnormal application using permissions as features. To compare results, we use the same 'Photography' application with the injected malware dataset. We build binary vectors for all applications and perform Naive Bayesian classification to identify risky applications. The performance of binary vector in represented on Figure 6.4.4 as a dark-purple dotted line.

The bold lines in Figure 6.4.4 show the ROC curves for our methodology performance for beta from 0 to 4. To simplify the comparison, we provide the performance of our method for different β always keeping LN_0 equal to 0. The purple line corresponding to the $\beta = 0$ is inverted, as the original values were below the random. This means that in the case of $\beta = 0$, the best performance is obtained when applications with a privacy score higher than a threshold are considered as risky. This happens due to the fact that many risky applications require many permissions. In spite of the inversion, the results are low. $\beta = 1$ represents a linear separation and shows better results, but the best results are obtained for $\beta = 3$. The Figure 6.4.4 shows that the risk score and warning threshold proposed in this paper becomes much more accurate than other approaches starting from the $\beta = 2$. Even if the approach presented in [66] outperform the binary vector, it is outperformed by our method with the $\beta = 3$. This comparison justifies the necessity of the adjustment parameter β in the equation 6.3.7. From the results, we conclude that that the hypotheses 1 and 3 of this study are confirmed and the answer for the second research question is positive. To obtain the final warning system, the best β, *threshold* and LN_0 should be chosen in the same way for each category.

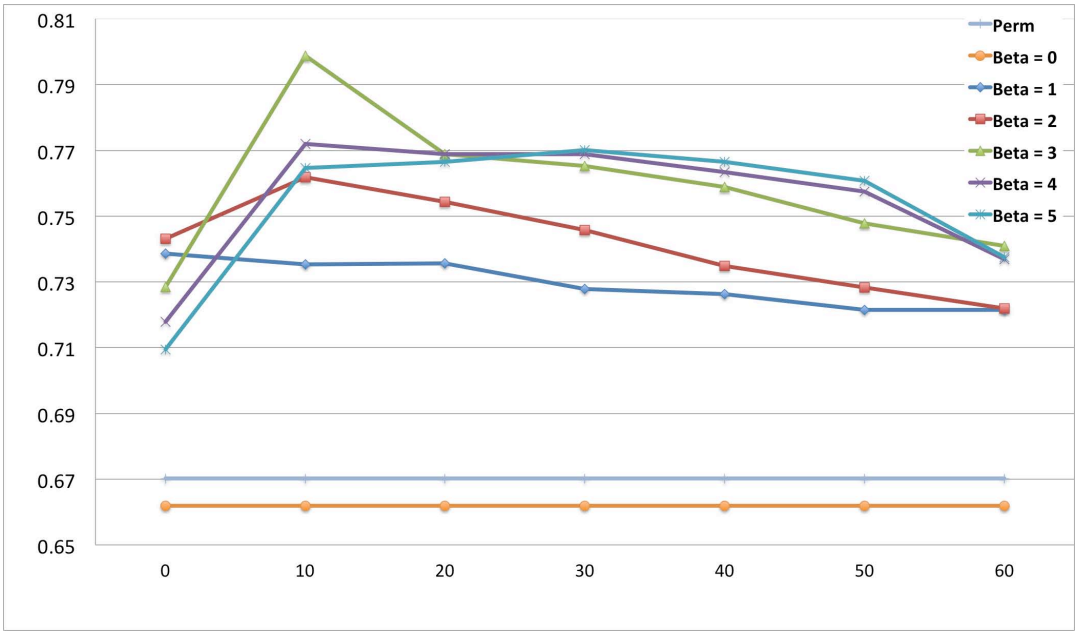

Figure 6.4.5: Graph representing how the minimal likeness and beta affects risky application detection.

Beta	F-measure	F_2-measure	$F_{0.5}$-measure	Min Likeness
1	0.738	0.640	0.862	0
2	0.761	*0.568*	0.860	10
3	**0.798**	0.658	0.872	10
4	0.772	0.692	**0.874**	10
5	0.770	0.690	**0.874**	30
0	*0.661*	0.679	*0.595*	150
Perm	0.670	**0.701**	0.780	N/A

Tableau 6.8: Best results for F-measure, F2-measure and F0.5-measure for different β

6.4.5 Category similarity

A confusion matrix obtained by a Naive Bayesian classification of applications into categories using all graph features shows the original and the predicted class for the data entries. Those data allows the creation of a category similarity graph by linking each category with a wrongly predicted category. The aim of this representation is to highlight the categories that are the hardest to predict and closest to other categories. Figure 6.4.7 shows the category similarity graph.

Some categories contain similar types of applications: 'media and video' with 'entertainment'; 'travel and local' with 'transportation'; 'app wallpaper' and 'personalization'; 'communication' and 'tools'; 'productivity' and 'business'; 'finance' with 'books and references'; 'sports' with 'health and fitness'; games related categories and, surprisingly, books and medical.

The applications are assigned into categories by developers and there are no rules for assigning an application to a category. Some categories have similar names, and develop-

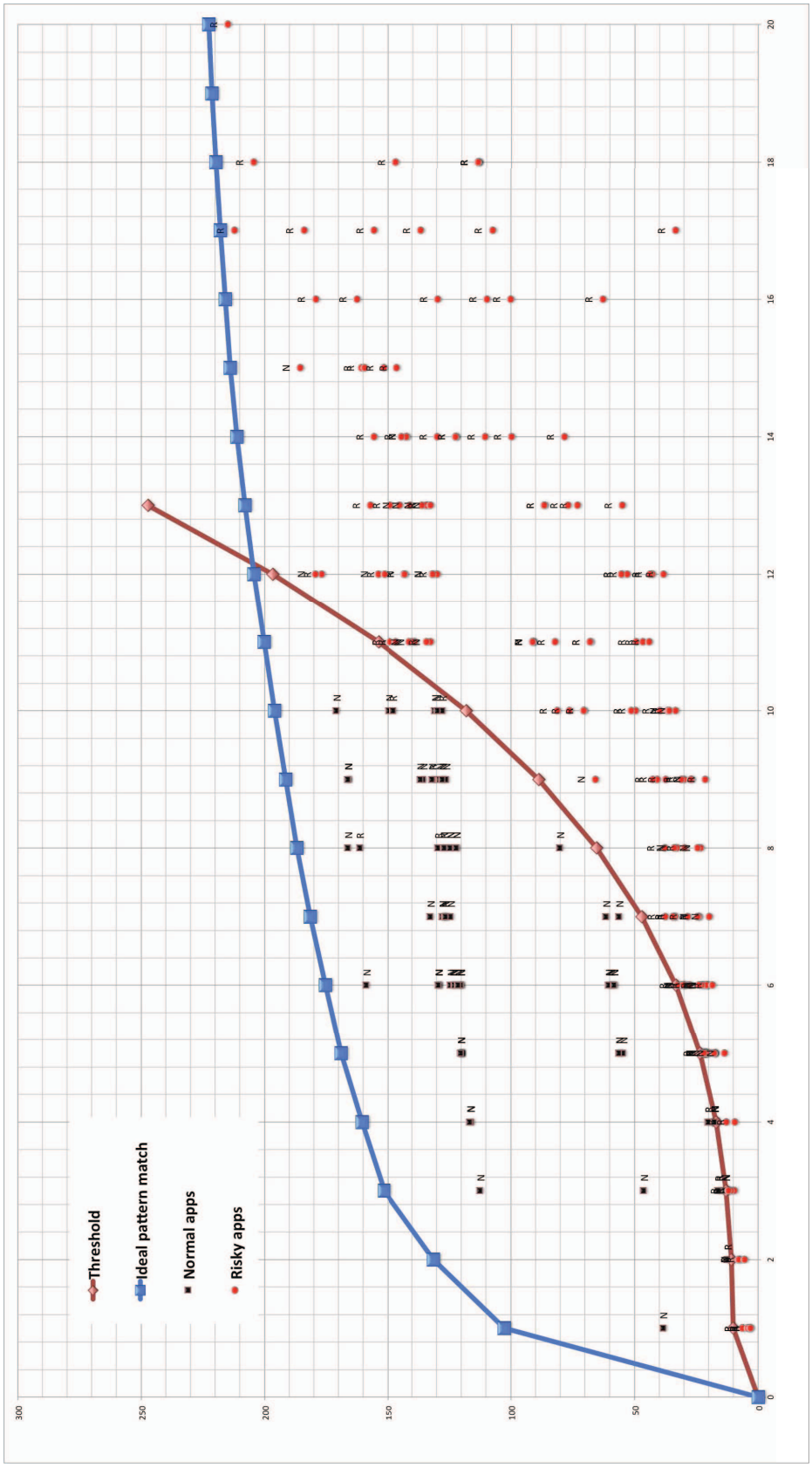

Figure 6.4.6: Optimal threshold for detecting risky applications $\beta = 3$, $threshold = 0.108$, $LN_0 = 10$

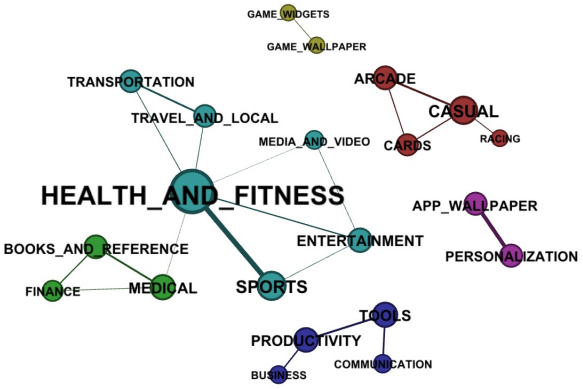

Figure 6.4.7: Category similarity graph

ers assign up to three categories to one application to make the applications more visible: for example, wallpaper applications can also be found in the 'personalization' category. A few developers assign only the best category. This can lead to false positives between similar categories, and reduce the automatic classification results. Some categories are large and contain different types of applications. For example, the 'photography' category includes applications with different functionalities: camera, image processing and combinations. Some camera applications enable users to take photos, such as simple camera, panoramic photo and heat photo. Most photography applications are photo processing apps, including an actual camera or otherwise, providing photo filters, adjustment settings, photo frames, custom text, photo montage, etc. Finally, a few applications were concerned with secure photo management. Under the hypothesis, normal applications should use permissions found centrally in the pattern, but some image processing applications do not actually require camera access. Android does not include a permission to access the photo gallery that would be central to both functionalities, therefore more fine-grained and data oriented permissions would be beneficial for the proposed methodology. Google Play could also integrate more fine-grained and explicative categories and subcategories, and clearly define what kind of application goes into each category. That would be a benefit not only for automatic analysis, but also for users looking for specific applications.

6.5 Discussion and future work

The pattern construction is based on the dataset, therefore it should be carefully prepared and filtered. Google Play is supposed to have a majority of normal applications but to ensure the quality of the pattern, in addition to the filters explained in the proposed methodology, one can choose to consider only applications with a high rating/user note. We also observed that some developers produce many similar applications, therefore the dataset can be filtered to omit over-productive developers or to limit the number of applications created by the same developer by category to ensure the diversity.

The proposed methodology treats all permissions that deviate from the pattern equally, while a person may consider some permissions more risky. Known risks and the additional penalty for dangerous permissions that are required but not in the pattern can be integrated into the risk measurement.

Compared to data flow tools described in the state-of-the-art section, the proposed solution is lightweight and based entirely on easily accessible data. Code analysis is a complex and heavy task but those data flow analysis-related works could be complementary to the proposed approach. If the user is willing to control applications' data flows, he could choose to run tools to analyse and control particular risky application instead of monitoring all applications and data flows. Those tools could also be used as a preprocessing step to the proposed methodology for choosing only permissions that are really used and not merely required, but it would add additional complexity to the methodology (for example, those unused permissions should be removed from the application itself to avoid permission escalation attacks).

This work can be continued by introducing an application recommendation system that not only takes into account application ratings, but also privacy, using a normal permissions request pattern. The proposed methodology allows us to measure risk of an application using two axes: number of permissions and pattern likeness. Thereby, the future recommendation system could propose applications that are similar to the one chosen by the user, but having less risk. This can be rather an application requiring less permissions for similar likeness, an application having a maximum likeness for the same number of permissions or a balanced recommendation.

6.6 Conclusion

Permission system is one of the Android security mechanisms. A list of permissions required by an application is supposed to warn users about abnormal or risky applications, but the state-of-the-art showed it is ineffective. We suppose that permission list is an important source of information that can be analysed automatically to propose a simpler and more effective security indicator for users.

We observe that Google Play groups Android applications into categories. We suppose that applications would be similar within a category and would require similar permissions, but applications from different categories would have different permission requests. Therefore, we could identify permissions patterns that characterise each category and a typical application of each category. Knowing normal behaviour of an application of a given category, we can identify applications that deviates from it: wrongly categorised, poor, abusive or malicious. This study verifies the aforementioned hypothesis and concludes that categories can be described by permission patterns, and a risk warning based on such patterns allows malware detection.

In this study, we model normal permissions requests by category using graphs. We propose a 5 step methodology to obtain patterns for each category. We verify the performance of different patterns in application classification into categories to select the most descriptive patterns. Betweenness centrality and weighted degree patterns were identified as the best-performing for classification. When combined, they provide the highest gain in application classification performance and also allow to consider each permissions of the pattern. We, then, propose a privacy score based on the similarity between an application and a given category-pattern and a risk warning threshold. We test our risk warning in malware detection and it outperformed the similar works from the state-of-the-art.

The proposed study address several issues of the state-of-the-art. First, our methodology is based on the information that can be easily obtained without using any complex code analyses techniques. Second, out risk score is based on the expected behaviour of benign applications in different categories, therefore the user could be warned about any

abnormal permissions request and not only about the known malware. Third, the pattern construction methodology is separated from the risk measure. This way, the risk of any application in any category can be easily calculated without the need of rebuilding patterns often. The same methodology could allow to chose the best category for new applications arriving to the market.

The behavioral pattern explained in this work was presented at The Seventh International Conference on Pervasive Patterns and Applications PATTERNS 2015 [133]. The full version of the study is under revision for the Decision Support System Journal, Elsevier.

Chapter 7

Respecting user's privacy by default: a PbD permission system for mobile applications

Abstract

The Privacy by Design concept proposes integrating respect for user privacy into systems managing user data from the early design stage. This concept has increased in popularity and the European Union (EU) is enforcing it with a Data Protection Directive. Mobile applications have entered the market and the current law and future directive are applicable to all mobile applications designed for the use in EU. It has so far been shown that mobile applications do not suit Privacy by Design and lack transparency, consent and security. Current permission systems are judged unclear for users. This chapter introduces a novel permission system suitable for mobile application that respects Privacy by Design. Instead of predefined permissions, this chapter presents a permission system vocabulary permitting the generation of permissions necessary for each piece of personal data. The patterns for the definition of permissions and also for enforcement implementation are given. Such an adapted permission system can improve not only the transparency and consent, but also the security of mobile applications. Finally, an example of a permission definition for a real mobile application using a lot of personal data illustrates the model.

7.1 Introduction

Mobile phones have significant data flow: information can be received, stored, accessed and sent by applications. Data can be entered by the user, retrieved from the system sensors or applications, come from another mobile application, arrive from servers or from other devices. Data can be shared on the phone with another application, with servers or other devices.

The 'Opinion 02/2013 on apps on smart devices' by the Article 29 Data Protection Working Party [8] defines four main problems with mobile privacy: lack of transparency, lack of consent, poor security and disregard for purpose limitation.

One of the security mechanisms protecting data in mobile and other systems is the Data Access Model or a permission system. Permission systems are embedded in mobile operating systems and are a crucial part of mobile security and privacy. Well designed, it makes a proactive privacy-respecting tool embedded in the system. Nowadays, permission systems do not follow the Privacy by Design principles: a number of studies concluded that the Android permission system fails to provide transparency, consent, purpose limitation and security [39, 40, 38, 42]. Users cannot understand Android permissions and the related functionalities or risks, ignore repetitive notifications and cannot control data access after application installation. A redesigned permission system could address all four problems of mobile privacy outlined earlier.

This chapter presents the permission system model and vocabulary where permissions can be built for each piece of private data, each action and for each purpose. Such a system can cope not only with security and control problems, but also with the transparency, consent and purpose-disregard problems. The remainder of the chapter is organized as follows: Section II summarizes state-of-the-art studies and highlights the limits of related works. Section III introduces the privacy-respecting permission system model: an overview, the vocabulary and permission states. Section IV presents the permission system in action: algorithms to define and to enforce permissions are included in this section. Section V shows the application of the novel permission system to a real mobile application. The paper ends with discussion, future work and conclusion.

7.2 Related works and limits

Many works proposed improvement to the Android permission system. Authors of [67, 68, 69, 63] proposed tools enabling the possibility of revoking permissions mostly by returning the shaded data and authors of [71, 72, 74, 73] presented condition-based granting tools: whitelist applications, context-based rules or user-defined rules such as the number of times used and time of the day the permission can be used. It is not clear if users can adopt those technologies as no studies have been done on the subject yet. It is known that users do not understand many of the Android permissions and its purposes, therefore users are not likely to be able to configure the current permission system, especially at installation time. Moreover, the revocation of permission can break many applications that rely on them and do not expect to be thrown an exception. The authors of [94, 96, 97] propose dynamic analysis tools monitoring the data and preventing private data leakage. Although such a system add transfer protection to the native access protection, it is not clear if users can configure such tool. The authors of [78] also propose adding transfer permissions linking data-related permissions to transfer-related permissions such as 'contacts -> sms'

or 'contacts -> Internet'. One can see that the presented tools tries to cope with the control and security problems, but transparency and user's consent seems limited.

The authors of [80] propose controlling each method call of Objective-C. This permission mechanism is clearly fine-grained, but seems even more complex for users to adopt than actual permission systems.

In [77], the authors proposed integrating new fine-grained permissions to restrict internet access to a particular domain, clarify publicity-related permissions, anonymize and reduce some permissions. The authors of [79] propose creating permissions not only for an entire SQLite table, but also for certain lines, columns or cells. Although those works considerably improve the permission system, it still remains incomplete.

The authors of [75, 76] proposed digital right management model and its implementation adding constraints and restrictions to the file usage. Such works make per-file permission systems and only protects the access action. The goal of such a system is to protect files from unauthorized usage and distribution rather than to protect users.

To our knowledge no work has been conducted on redefining the mobile permission system to fit the Privacy by Design principles or on adding purpose to permissions.

7.3 Privacy-respecting permission system overview and vocabulary

This study proposes to focus on permissions concerning data and the actions that can be carried out on this data, rather than on the technology used. The definition of purpose of the data use is also included in the proposed permission system.

Privacy Policy should be short and clear. Users should have a global vision of the data use and functionalities before they install an application. Users rarely read long involved policies, especially when they want a service and feel they have no choice but to accept all permissions. The proposed permission system enables a simple policy to be generated with a list of permissions.

7.3.1 Definition

The permission system is modeled on an access control model. Discretionary access control is chosen where only the data owner can grant access. The user should be able to control the data, therefore, the user is considered as the unique owner of all information related to him.

$\mathcal{R}_{app}$ is a set of *rules* assigned to the application *app*. A *rule* is defined as an assignment of the $\mathcal{R}ight$ over an $\mathcal{O}bject$ to a $\mathcal{S}ubject$. The *rule* triplet is defined as follows:

$$\forall\, rule \in \mathcal{R}_{app},\, rule = (s,\, r,\, o) \tag{7.3.1}$$

where $s = \mathcal{S}ubject$, $r \in \mathcal{R}ight$, $o \in \mathcal{O}bjects$

A mobile application is defined as a $\mathcal{S}ubject$. Each mobile application is associated with the unique identification number that can be used as a $\mathcal{S}ubject$.

$$\mathcal{S}ubject = Mobile\,Application\,ID \tag{7.3.2}$$

$\mathcal{O}bjects$ are the user-related data, such as e-mail, contact list, name and surname, phone number, address, social networks friend list, etc. As the permission system is data-

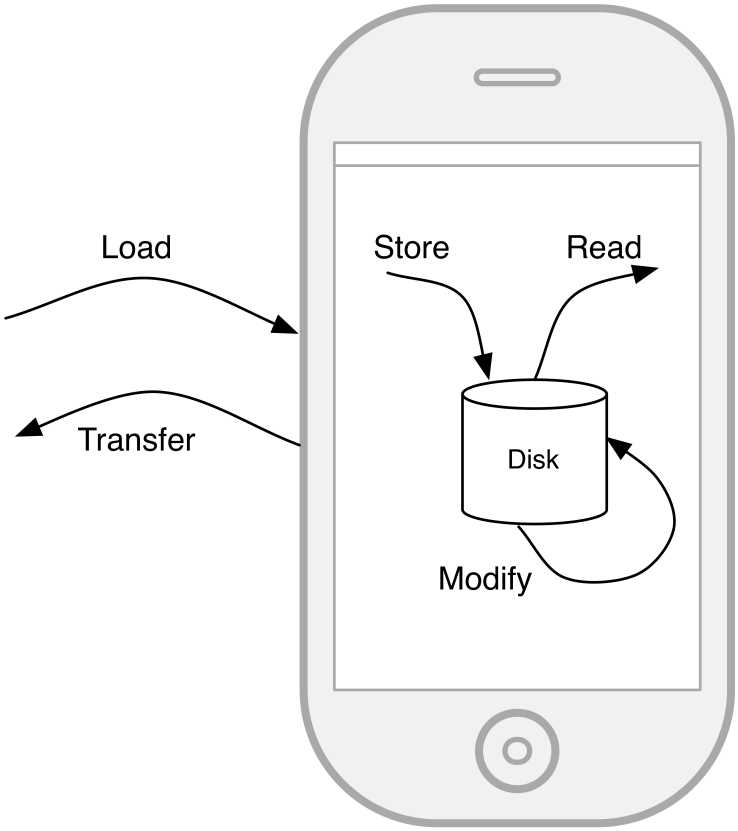

Figure 7.3.1: Different actions can be made on the data and included into the proposed permission system.

centered, the definition of data should be as precise as possible.

$$\mathcal{O}bjects = \{Phone\#, Name, Contacts, \cdots\} \tag{7.3.3}$$

Each application needs to use a piece of personal data to perform a particular action with a specific goal. Users give the $\mathcal{R}ight$ to the application according to this action and this goal. To define $\mathcal{R}ight$ we have to introduce $\mathcal{A}ction$ and $\mathcal{P}urpose$.

Each action is one of all the actions, denoted $\mathcal{A}ctions$, that can be carried out on user private data by the application: load, read, modify, store and transfer. We define the $\mathcal{A}ctions$ as follows:

$$\mathcal{A}ctions = \{Read, Modify, Load, Store, Transfer\} \tag{7.3.4}$$

where

$\mathcal{R}ead$ is a read-only access to the data that is already stored on the phone.

$\mathcal{M}odify$ is an action permitting the update of a piece of personal data already stored in the system.

$\mathcal{L}oad$ represents an action bringing new information to the phone from a distant server, Internet or mobile sensor such as GPS, etc.

$\mathcal{S}tore$ action indicates a new piece of private data will be saved on the device.

$\mathcal{T}ransfer$ action indicates some private data is transmitted from the device to the server or another device.

Actions are depicted on Figure 7.3.1.

$\mathcal{P}urpose$ is assigned by the application developer and depends on the service. For example, purpose could be 'retrieve forgotten password', 'display on the screen', 'calculate the score', 'send news', 'retrieve nearest restaurant' and 'attach to the message'.

$$\mathcal{P}urpose = \{Retrieve\, forgotten\, password, \cdots\} \tag{7.3.5}$$

A permission right, denoted $\mathcal{R}ight$, for all actions except the $Store$ action is defined as a combination of one element of $\mathcal{A}ctions$ and one element of $\mathcal{P}urposes$.

To respect the minimisation principle (minimize the data usage and storage), any personal data should be stored on a mobile device only the period of time that is necessary for the functionality. The set of rights, denoted $\mathcal{R}ight$, with the action equal to $Store$ is defined with an additional parameter, $time$, informing about the time storage.

$$\forall r \in \mathcal{R}ight,\ r = \begin{cases} (action,\ purpose) & if\ action \in \mathcal{A}ctions - \{Store\} \\ (action,\ purpose,\ time) & if\ action = Store \end{cases} \tag{7.3.6}$$

where $purpose \in \mathcal{P}urposes$, $time \in [0, T]$. The period $[0, T]$ is an application lifetime.

The time storage can indicate the number of days, hours or months data is stored or the time regarding the application lifecycle: until the application is closed, until the application is stopped, until the application is uninstalled. All personal data available during the removal of the application is deleted regardless of the defined period, as it cannot exceed the application lifetime.

Figure 7.3.2 summarizes the definition explained above.

7.3.2 Object: private data

Existing mobile permissions and mobile forensic techniques on iOS and Android phones are allowed to identify some private data that can be accessed on the smartphone by an

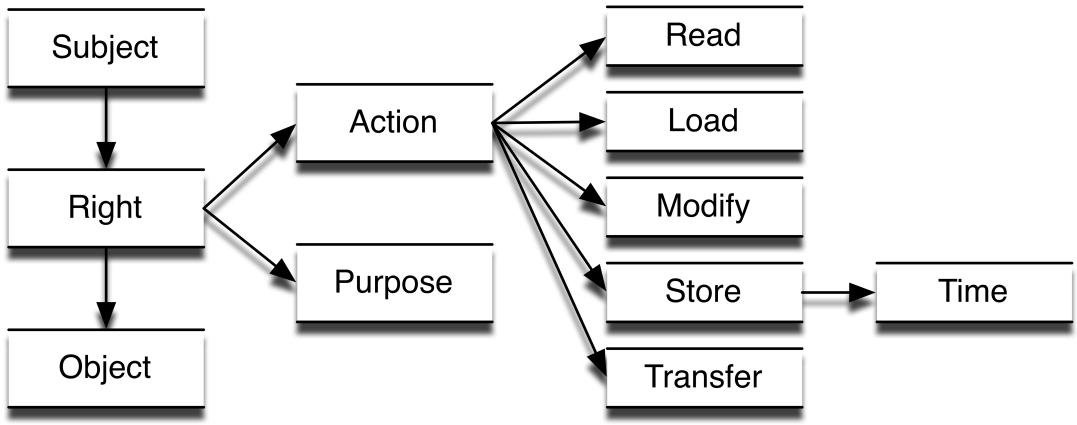

Figure 7.3.2: Summary of the the proposed permission system definition

application. Below is a, possibly not exhaustive but, large list of personal information that can be found on a mobile phone.

- Contact list: type (phone, Facebook, Twitter, Skype) or associated service names; name; surname; nickname; e-mail; picture; address; website; company; birthday; job title; significant other
- Calendar: name; appointments containing subject, location, starting date, ending date, status, notes, attendees' full names.
- SMS / MMS: type (incoming, outgoing); time; sender's name; sender's phone number; text content; multimedia content (SMS: small images; MMS: images, photos, video, contact information, etc.)
- Call logs: type (incoming, outgoing, missed); caller's name, caller's phone number; voice messages.
- Location: exact location (latitude and longitude); approximate location (latitude and longitude), address, street, city, country.
- Stored Multimedia: type (image, audio, video); multimedia content; META data (date, geolocation).
- Medical records: weight, height, sex, body mass index, nutrition, illnesses, blood pressure, heart rate, sleep quality, sexual activities, etc.
- Other: multimedia sensors (newly created image, audio, video) and data from other sensors (pressure, temperature, movement, etc.); mobile phone usage statistics (last used applications, configurations); device unique id; SIM id; web browser history; user accounts information (login, password, tokens); documents from external or internal storages; currently displayed screen (screen shots), push messages; bank account information; biometric information; sport activities; social networks and other mobile applications' data; list of installed applications.

Android includes only 26 permissions giving access to personal data including location, accounts, SMS, MMS, camera, audio and video content, mobile user activities, calls,

contacts, calendar, saved documents, external storage data and screen shots. The Android 'Personal info' permission group contains only 16 read and write permissions where only 5 permissions give read access to personal data. Other permissions are distributed between 'Location', 'Messages' and 'Hardware control' permission groups. One can see that very few permissions exist on Android to protect personal data, compared to the large amount of personal data that can be available on a mobile device.

Note, that the $o \in \mathcal{O}bjects$ can represent data of different granularity. For example, the permission can be assigned to the 'Web browser history', or to 'Today's Web browser history'; to 'Contact list', to 'Business contact list' or 'Personal Contact's Names'; to 'Images', 'Application-created images', 'Image gallery folder' or to 'JPG images'; 'latitude and longitude', ' approximate latitude and longitude', 'Street', 'City' or 'Country', etc. In the filesystem, the $o \in \mathcal{O}bjects$ the permission is assigned to can represent a folder, file or a certain type of files, to the database, database table or certain lines, columns or cells.

7.3.3 Permission use restrictions

To reduce the flexibility of permission usage by an application and to give more control to the user, the Privacy by Design permission system proposed in this study assign several simple restrictions to each rule. Each permission is associated with permission restrictions under which the user accepts the permission. Restrictions should be simple for the user to set up.

Each $\mathcal{R}estriction$ contains an action type from the set of action types denoted $\mathcal{A}ctionTypes$. Two action types are defined: an action that will be explicitly launched by the user, denoted $\mathcal{U}serEvent$, and automatic action, denoted $\mathcal{A}utomatic$.

$$\mathcal{A}ctionTypes = \{UserEvent, Automatic\} \tag{7.3.7}$$

$\mathcal{U}serEvent$ action is launched via the user interface by the user triggering a particular event. $\mathcal{A}utomatic$ action is launched by an application and can include regular access, update and transfer of the data or automatic insertion of a complementary piece of data (e.g., automatic attachment of the geolocation data to the message, automatically fill in the form, automatically synchronise the data with the server, etc.). $\mathcal{A}utomatic$ action can be triggered without any action by the user.

$\mathcal{U}serEvent$ restrictions attach the permission to a particular user event. We define $\mathcal{R}estriction$ for the $\mathcal{U}serEvent$ action type as follows:

$$\forall res \in \mathcal{R}estriction,$$
$$res = (rule, action\_type, user\_trigger\_event) \tag{7.3.8}$$

where $action\_type = UserEvent$, $rule \in \mathcal{R}_{app}$, $action\_type = UserEvent$ and $user\_trigger\_event$ is a concrete user event intercepted by an application.

$\mathcal{A}utomatic$ action can be triggered by the system with a certain frequency or can be associated with a trigger event (e.g., send message, create new message, etc.) or both. We define $\mathcal{R}estriction$ for the $\mathcal{A}utomatic$ action type as follows:

$$\forall res \in \mathcal{R}estriction,$$
$$res = (rule, action\_type, frequency, event) \tag{7.3.9}$$

where $action\_type = Automatic$, $rule \in \mathcal{R}_{app}$, $frequency$ represents the number of times per day/week/month the action can be performed by an application; *event* is an event associated to the action launch: application is opened, screen is shown, message is sent (button is clicked), form is filled in, etc.

$Frequency$ is not a mandatory parameter. One permission can have several restrictions: it can be associated with several different user or application events.

7.3.4 Permission state

The user should be explicitly asked to assign each *rule*. Thus, each *rule* has a $\mathcal{S}tate$: granted or revoked. The rule is granted with corresponding restrictions or the rule is revoked entirely. To respect the 'Privacy by Default' notion of Privacy by Design, the default $\mathcal{S}tate$ of the permission in installed applications is *Revoked*. Only the user can modify the $\mathcal{S}tate$ of permission with an explicit action via the user interface.

The $\mathcal{S}tate$ is defined as follows:

$$\mathcal{S}tate(rule, time) = \begin{cases} Granted, & user\,accepts\,the\,rule \\ Revoked, & user\,declines\,the\,rule \end{cases}$$
$$\mathcal{S}tate(rule, 0) = Revoked \tag{7.3.10}$$

where $rule \in \mathcal{R}_{app}, time \in]0, T]$.

The *State* of a rule $r1 \in \mathcal{R}_{app}$ changes over the application lifetime. The diagram in Figure 7.3.3 shows an example of state modification.

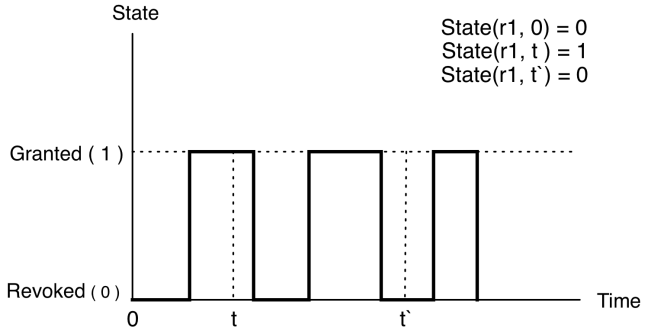

Figure 7.3.3: Example of state modification diagram for a given permission

7.3.5 User control

User should have a choice of granting the permission permanently, of revoking the permission permanently or of confirming the permission usage with the user each time. The *rule Check* is defined as follows.

$$\mathcal{C}heck(rule, time) = \begin{cases} True, & confirmation\,required \\ False, & no\,confirmation \end{cases}$$
$$\mathcal{C}heck(rule, 0) = True \tag{7.3.11}$$

where $rule \in \mathcal{R}_{app}$, $time \in]0, T]$

If the $\mathcal{C}heck$ parameter is set to $True$, than the $\mathcal{S}tate$ passes to *Revoked* and is ignored by the system. The permission is granted or revoked each time by the user via the user interface allowing execution of the functionality, thereby the $\mathcal{R}estriction$ of the permissions is also ignored.

If the $\mathcal{C}heck$ parameter is set to $False$, the system verifies the permission $\mathcal{S}tate$ and $\mathcal{R}estriction$ in order to execute the functionality.

To respect the 'Privacy by Default' notion, the default $\mathcal{C}heck$ parameter is set to $True$.

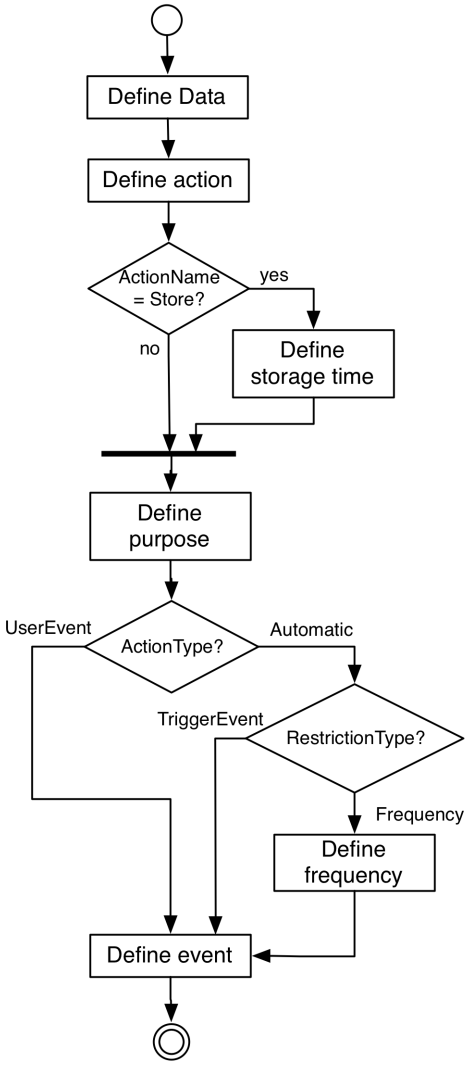

Figure 7.3.4: Activity diagram for the rule definition

7.3.6 Permissions interconnection

Each permission is associated with the purpose, thereby each permission is associated with one particular functionality. Several permissions may be needed to assure one functionality, and the developer can give the user a choice of using one permission or another. Several permissions can be grouped by functionality in two ways: all permissions are needed to assure the functionality; only one permission is necessary to assure the functionality.

The $\mathcal{G}roupType$ parameter is defined as follows:

$$GroupType = \{\mathcal{A}LL, \mathcal{O}NE\} \tag{7.3.12}$$

where $\mathcal{A}LL$ shows all permissions are necessary to assure the functionality. $\mathcal{O}NE$ shows only one of the listed permissions is necessary to achieve the functionality.

The $\mathcal{A}LL$ parameter can be expressed as a single permission that should be accepted or declined by the user or by one activation button grouping all permissions. To respect the minimisation principle, all permissions linked to a particular functionality should be *Revoked* if at least one permission was declined by the final user.

The $\mathcal{O}NE$ parameter can be expressed by the user interface as a group of radio buttons or as one permission with a drop-down list(s) on parameters that differ from permission to permission (e.g., data, usage frequency). If at least one permission is given by the user, the corresponding functionality will be assured and other permissions will be Revoked. For example, an application can assure the service with different types of geolocation data: latitude and longitude, city and the street name and the city only. Developers can propose that the user choose one of the types of geolocation data.

Several permissions can be grouped to add dependencies and an acceptance rule. We define the $\mathcal{G}roup$ parameter as follows:

$$\forall group \in \mathcal{G}roup, group = (group\_type, \{rule_x, rule_y, \cdots\}) \tag{7.3.13}$$

where $group\_type \in \mathcal{G}roupType$; $rule_x \in \mathcal{R}_{app}$; $rule_y \in \mathcal{R}_{app}$

7.4 The permission system in action

The developer should define the permissions for all personal data ($\mathcal{O}bject$) used in the application ($\mathcal{S}ubject$) before making the application available on the market. Permission restrictions and permission groups should also be defined. Figure 7.3.4 shows the recapitulative schema of the permission definition.

The permission ($rule$) is stored inside the application with its current $State$ and $Check$ parameters. The default $State$ is *Revoked*. The default $Check$ is $True$. Developers should verify that the permission is fully *displayed* with the corresponding object, action, purpose and restrictions and *requested* at least once and that the user is able to grant or revoke this permission. Finally, the user should stipulate the settings with all $rule \in \mathcal{R}_{app}$ to be able to *Grant* or to *Revoke* individual permissions later.

The activity diagram in Figure 7.3.5 shows the permission management cycle from the permission request to the permission usage authorisation/non-authorisation. When one permission is required by the application, the permission management system first verifies the parameter $Check$. If $Check$ is set to $True$ the system generates the message including all information about the required permission. Users can accept or decline the permission as well as switch permission management to automatic (Set $Check$ to $False$). An example of such message is shown in Figure 7.4.1. According to the user answer the permission is authorised or non-authorised. $Check$ is set to $False$ meaning an automatic permission management is enabled. System verifies the permission's *Status*. If permission *Status* is *Revoked*, then the permission is non-authorised. If permission *Status* is *Granted*, then the systems check corresponding *Restrictions*. If *Restrictions* are respected, the permission is authorised, otherwise the permission is non-authorised.

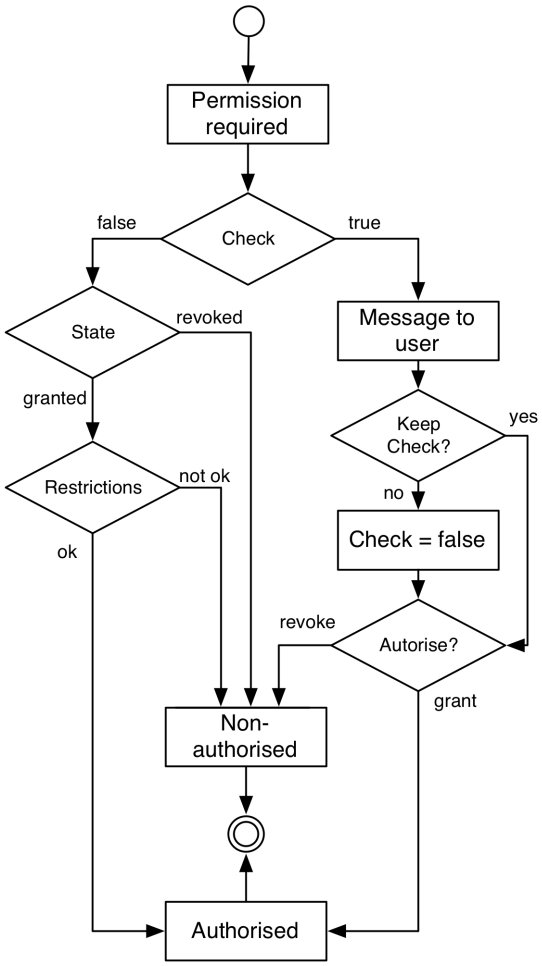

Figure 7.3.5: Activity diagram: permission management.

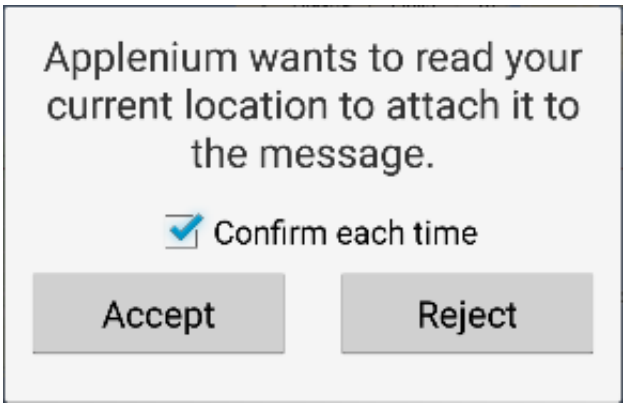

Figure 7.4.1: Example of a permission request

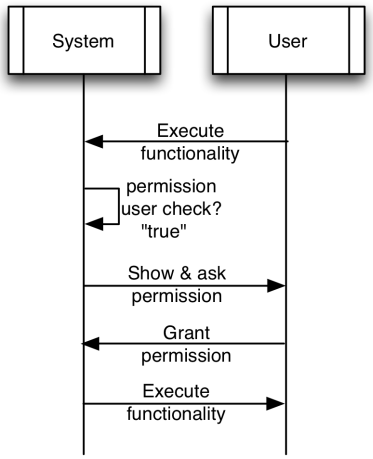

Figure 7.4.2: Sequence diagram: first use of one permission or a use or permission in 'user check' mode.

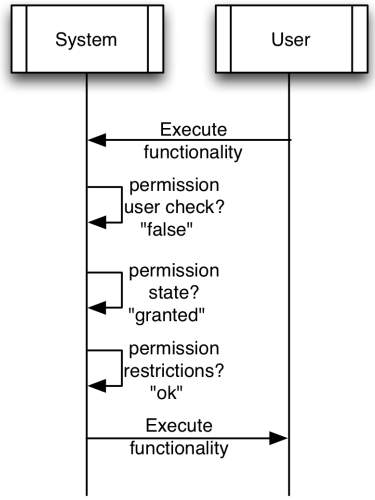

Figure 7.4.3: Sequence diagram: use of one permission without user confirmation.

The sequence diagram in Figure 7.4.2 shows the case of permission management when the permission is used for the first time or the *Check* parameter is set to $True$. For example, the user wants to invite a friend to play a game together, the application needs permission to access the list of contacts and full name with emails to send an invitation mail. System verifies the parameter *Check* that is set to 'true'. System generates the message for the user explaining the permission needed, the user accepts the permission. Now user can choose the friend he wants to invite.

The sequence diagram in Figure 7.4.3 shows the patterns in action when the permission is used automatically without user confirmation: *Check* parameter is set to $False$. Permission that does not require user confirmation can be used by an application if, and only, the permission status is $\mathcal{G}ranted$ and all $\mathcal{R}estrictions$ are respected. For example, the user wants to automatically attach his geolocation (city) to the first message it sends per day. The permission to load the city from the GPS is needed to attach it to the message. Permission is restricted to a particular event - create new message - and a frequency, once per day. When the user creates a new message, the system verifies the *Status* of the described permission. As *Status* is granted, the system verifies the *Restriction* - permission was not used yet today, therefore, the permission use is authorised.

One can see that the time in which permission can be used by an application is shorter than the time in which permission is Granted. $UserEvent$ action-type permissions can only be used following a particular user event, therefore, permission becomes active only at a specific time. *Automatic* permissions are limited by the defined frequency or an event. The usage diagram is depicted in Figure 7.4.4

7.5 Application Example

This section gives an example of the permission system made for the application of trust evaluation of friends on social networks named Socializer 1.0 [134]. This application was

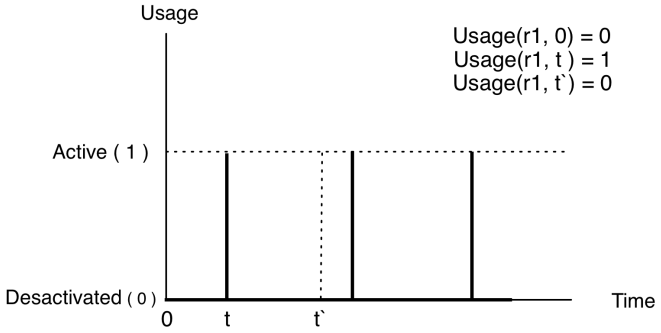

Figure 7.4.4: Example of usage modification diagram for a given permission

chosen because its service is based on private information and cannot be anonymous, the Privacy by Design notion should be integrated into this application.

This application needs user friend lists of different social networks (Facebook, Twitter, LinkedIn) plus the contact list to view friends and mutual friends to calculate the overlap of friends in different social networks and contact lists and to evaluate the trust of Facebook friends.

The following private data can be used by the application:

1. List of contacts from mobile address book: name, surname
2. Facebook friends list: name, surname
3. List of mutual Facebook friends for each Facebook friend: name, surname
4. Twitter friends list: Name, Surname
5. Daily Facebook messages for each Facebook friend
6. Calculated trust score

Contact list is found on the smartphone, therefore, the application needs a *Read* right.

$$rule_1 = (s,\ (Read, purpose_1),\ ContactList)$$

where s is a *Subject* defined as the application Socializer 1.0; $purpose_1$ = calculate the trust scores.

The application will load the user contact list on user event: onClick on the button "load contact list".

$$restriction_1 = (rule_1, UserEvent,\ event_1)$$

where $event_1$ is a user event: onClick on the button 'load contact list';

Social networking friends lists should usually be retrieved from the server of a given social network, therefore, the load and store actions should be defined. The Facebook friends list plus the contact list are essential to assure the overlap and trust functionality.

$$rule_2 = (s,\ (Load, purpose_1),\ FacebookFriendList)$$

$$rule_3 = (s,\ (Store, purpose_1, time_1),\ FacebookFriendList)$$

$$restriction_2 = (rule_2, UserEvent,\ event_2)$$

$$restriction_3 = (rule_3, UserEvent,\ event_2)$$

where $time_1$ is a storage time defined as: while the application is installed; $event_2$ is a user event: onClick on the button "load Facebook friends list";

For each Facebook friend, the list of mutual friends with the user is necessary for trust calculation.

$$rule_4 = (s,\, purpose_1), FacebookMutualFriendLists)$$

$$restriction_4 = (rule_4, UserEvent,\, event_3)$$

where $event_3$ is a user event: onItemClick on the user name in the list of users for trust calculation;

A list of friends from other social networks improves the scores of overlap and trust.

$$rule_5 = (s,\, (Load, purpose_2),\, TwitterFriendList)$$

$$rule_6 = (s,\, (Store, purpose_2, time_1),\, TwitterFriendList)$$

$$restriction_5 = (rule_5, UserEvent,\, event_4)$$

$$restriction_6 = (rule_6, UserEvent,\, event_4)$$

where $event_4$ is a user event: onClick on the button "load Twitter friends list"; $purpose_2$ = improve the trust score;

$$rule_7 = (s,\, (Load, purpose_2),\, LinkedInFriendList)$$

$$rule_8 = (s,\, (Store, purpose_2, time_1),\, LinkedInFriendList)$$

$$restriction_7 = (rule_7, UserEvent,\, event_5)$$

$$restriction_8 = (rule_8, UserEvent,\, event_5)$$

where $event_5$ is a user event: onClick on the button "load LinkedIn friends list";

The second functionality of the application is to evaluate the behaviour of Facebook and Twitter friends to indicate potentially dangerous contacts. The behaviour evaluation is calculated by analysing the messages published by the given friend over time. The application needs a permission to load messages.

$$rule_9 = (s,\, (Load, purpose_3),\, NewTwitterFriendMessages)$$

$$restriction_9 = (rule_9, Automatic,\, \emptyset,\, event_6)$$

$$restriction_9 = (rule_9, Automatic,\, f_1,\, event_7)$$

where $purpose_3$ is a purpose defined as 'calculate the Twitter friends behavior'; f_1 is an action frequency: once per day; $event_6$ is a user event: onSlideDown on the list of messages; e_7 is an application event: onApplicationStarted.

$$rule_{10} = (s,\, (Load, purpose_3),\, NewFacebookFriendMessages)$$

$$restriction_{10} = (rule_{10}, Automatic, \emptyset, event_6)$$

$$restriction_{10'} = (rule_{10}, Automatic,\, f_1,\, event_7)$$

where $purpose_3$ is a purpose defined as 'calculate the Facebook friends behaviour'.

The third functionality proposes viewing today's Facebook and Twitter messages on the screen for the user.

$$rule_{11} = (s,\ (Store, purpose_4, time_2), TodayTwitterFriendMessages)$$

$$restriction_{11} = (rule_{11}, Automatic,\ f_1\, event_7)$$

$$restriction_{11'} = (rule_{11}, Automatic, \emptyset, event_6)$$

where $purpose_4$ is a purpose defined as 'view today Twitter messages'; t_2 is a storage time defined as: one day.

$$rule_{12} = (s,\ (Store, purpose_5, time_2), TodayFacebookFriendMessages)$$

$$restriction_{12} = (rule_{12}, Automatic,\ f_1,\ event_7)$$

$$restriction_{12'} = (rule_{12}, Automatic, \emptyset, event_6)$$

where $purpose_5$ is a purpose defined as 'view today Facebook messages';

The user has the option of sharing the scores by posting new messages on Facebook and Twitter. The user can also contribute to research by sending the anonymized trust and behaviour statistics to the developer. Those actions should be taken with the user's express consent.

$$rule_{13} = (s,\ (Transfer, purpose_6), FacebookFriendTrustScore)$$

$$restriction_{13} = (rule_{13}, UserEvent,\ event_7)$$

where $purpose_6$ is a purpose defined as 'share results on Facebook'; $event_7$ is a user event: onClick on the button 'share'.

$$rule_{14} = (s,\ (Transfer, purpose_7), FacebookFriendTrustScore)$$

$$restriction_{14} = (rule_{14}, UserEvent,\ event_7)$$

where $purpose_7$ is a purpose defined as 'share results on Twitter'.

$$rule_{15} = (s,\ (Transfer, purpose_8),\ AnonymizedTrust)$$

$$rule_{16} = (s,\ (Transfer, purpose_8),\ AnonymizedBehavior)$$

$$restriction_{15} = (rule_{15}, UserEvent,\ event_8)$$

$$restriction_{16} = (rule_{16}, UserEvent,\ event_8)$$

where $purpose_8$ is a purpose defined as 'contribute to the improvement of the methodology'; $event_8$ is a user event: onClick on the button 'help research'.

The final application has 16 rules required by the application for full functionality.

$$\mathcal{R}_{app} = \{rule_1, rule_2, rule_3, \cdots, rule_{15}, rule_{16}\}$$

Those rules can be combined into groups. The rules $rule_1$, $rule_2$, $rule_3$ and $rule_4$ have a common purpose, all rules should be accepted to achieve the functionality mentioned in the purpose: 'calculate the trust'.

$$group_1 = (ALL,\ \{rule1, rule2, rule3, rule4\})$$

Object	Action	Purpose	#
Contacts list	Read	Calculate Trust	1
Facebook friends list	Load; Store		
Facebook mutual friends	Load		
Twitter friends list	Load; Store	Improve Trust	2
LinkedIn friends list			3
Twitter messages	Load	Tw. friends behaviour	4
Facebook messages		Fb. friends behaviour	5
Today Tw. messages	Store; (1 day)	View Tw. messages	6
Today Fb. messages		View Fb. messages	7
Trust score	Transfer	Publish to Twitter	8
		Publish to Facebook	9
Trust and behaviour	Transfer	Contribute to research	10

Tableau 7.1: Table recapitulating permissions needed for the application (last column is a permission group number)

Similarly, $rule_5$ should be grouped with $rule_6$ and $rule_7$ with $rule_8$.

$$group_2 = (ALL, \{rule5, rule6\})$$

$$group_3 = (ALL, \{rule7, rule8\})$$

The rules from $rule_9$ to $rule_{16}$ should be accepted one by one to achieve the aforementioned purpose (to achieve the functionality). Finally, we obtain 10 permissions to be added to the application and controllable by the user. Table 7.1 recapitulates permissions.

To compare with the actual permission systems, (a) iOS requires contact list, Facebook and Twitter access permissions. (b) Android requires 'internet', 'read_contacts' and 'get_accounts' access permissions. Facebook and Twitter connections are managed with APIs that require permissions to be declared on the platform, but the permission management will not be available by default for users in the mobile application. iOS permissions give a certain transparency to the user but Android permissions are vague.

We obtained more fine-grained control of the application and the data including permissions to all necessary personal data, actions carried out on this data and corresponding purposes. The recapitulation table (Table 7.1) clearly shows what data are used for what purpose. This kind of table can be added to the privacy policy to improve transparency.

7.6 Discussion and future work

The work presented in this chapter is proposed as a conclusion for the thesis research and is focused on the theoretical model and the objective of more transparent and privacy-respectful permission system. This study is a starting point that opens many directions for future researches.

The described system can be implemented as a middleware, for example, on Android. The implementation can be based on a semantic file system: tags attached to files would allow defining permissions for different types of data and different granularities. The

control can also be carry out on files and folders as it is currently made on desktop systems. The system API calls should also be performed through the middleware. The database-related permission implementation might need additional models to be able to implement and enforce permissions by tables, columns, raws and cells.

The implementation would help to test the applicability of a such system by developer and also to test the acceptance of such a system by users. The user interface is an important axe for the future research. The user interface implementation can be based on the mixed approach between the traditional pop-ups and tutorial-like screens. The interface has an important role in permission systems influencing the user understanding and adoption of the system but also the adoption of the system by developers.

The impact of new privacy-respective permission system on users and developers could be measured by conducting real-life experiments. The impact of integration of the new permission system on design and development time can be measured, as well as particular situations and difficulties in applying the pattern. We have an additional hypothesis that the explicative application with high transparency improves user experience and leads to more positive perception of the same application, therefore the use of our permission system benefits the application owner.

The proposed model can be merged with digital right management models and privacy policy definition frameworks, such as [135, 136], to obtain a more powerful system. Questions of secure storage of permissions should be addressed. Permission themselves are sensitive data and should be accessible and modifiable only by the system. Permissions can be defined and stored using semantics that are already used in the state-of-the-art studies for permission modeling and DRM (taxonomy languages, such as OWL).

7.7 Conclusion

This chapter introduced a Privacy by Design permission system for a mobile application. This permission system is data-oriented, thus, the final user can easily understand what personal data is involved. The system includes actions that are missing from current iOS and Android permission systems, such as load and transfer, that improve transparency of the application. The system also includes simple restrictions to better control data use.

The novelty is including the purpose of the data use in the permission system. This clear purpose will help users to understand better why the data is used and to judge whether this permission is needed. Purpose in permission also forces developers to apply the minimization principle: a developer cannot use the data if he cannot define the clear purpose of usage. The compulsory purpose definition should help guard against the abusive permission declaration 'in case'. Finally, purpose gives the user more fine-grained control, as he can allow the same data to be used for one functionality but not for another. It is important for our system to integrate clear purpose and not a vague explanation (e.g., 'measure the frequency of application utilization' instead of 'improve user experience').

PbD states that the user should have control over his data and have Privacy by Default, therefore, permissions used in the application are revoked by default. Users should be clearly informed and asked to grant permission. Moreover, users should keep control of permissions during all the application use time, therefore, permission settings must be available.

The proposed permission system helps developers to comply with the law: it defines what permissions the developer should add to the application, but in the current state it cannot ensure that all necessary permissions are really added. The proposed pattern

indicates to the developer what should be added to the application to be more transparent, but if the developer decides to transfer data without asking permission, the pattern allows this (even if it is against the European law). The system verifying and enforcing such permissions can be implemented to control dishonest developers. The privacy policy generated can give the first indication permitting the evaluation of whether the data use is reasonable and the purpose is clear. Manual verification of an application can show the anomaly in permission system use.

The study presented in this chapter was validated by the research community. A short version of the corresponding article was presented on the The Sixth International Conference on Pervasive Patterns and Applications PATTERNS 2014 international conference [137] and the extended version was published in International Journal On Advances in Security [138].

Chapter 8

Conclusion

8.1 Problematic

Privacy is a part of fundamental human rights. Nowadays, with rapid evolution of smart and connected devices, users could feel their privacy is not always respected. Smartphones collect massive amounts of the user's data and store precise information about the device owner at any time of the day.

In European Union, privacy is protected by the European Data Protection Directive 95/46/EC and upcoming unified European Data Protection Regulation. Any of data controller and data processor should integrate privacy principles of the Directive to be compliant with the law. Such principles are also often referred as «Privacy by Design». «Privacy by Design» is a concept proposed by Dr. Ann Cavoukian that make respect to the user's privacy an important and compulsory part of the software design phase. Seven principles promote proactivity, security, visibility and transparency, «Privacy by Default» and respect of the user's privacy can be integrated into all systems without giving up neither security no privacy. All notions are explicitly or implicitly integrated into the Directive 95/46/EC. Although Directive 95/46/EC and European Data Protection Regulation is applicable to mobile systems and mobile applications, developers do not always follow the imposed principles and systems and applications are rarely made «Privacy by Design». Developers do not always know how to translate those high-level principles into the «code» and «components» language and are not aware of the responsibility they have over the user's privacy. This thesis proposes a set of patterns that would give mobile system more Privacy by Design.

8.2 Contributions

The contribution of this thesis is threefold. First, we propose an architectural design pattern for Android application developers. Google do not define any architecture that Android developers could follow. As a result, the quality of an application strongly depends on developer's experience; bad architecture can lead to incomprehensible and unmaintainable code and is more susceptible to contain bugs and security breaches that could lead to the data leakage. A good basic architectural pattern is needed to simplify the software design phase for the developer. Therefore, the developer can concentrate on functionalities, security and privacy without resolving code and architectural issues again and again. We propose a basic architecture for Android that helps to obtain clear, reusable

code and a better quality application. This contribution enforces «Privacy embedded into design» and «End-to-end security» principles of «Privacy by Design» in mobile system. The architectural design patters structures the code simplifying the use of other design patterns and also helps to avoid bugs and security breaches due to the code clarity. This work was presented on The Fifth International Conferences on Pervasive Patterns and Applications PATTERNS 2013 where obtained the best paper award [116]. The extended version of the article was published in the International Journal On Advances in Software, volume 7 in 2014 [117].

Second, we observed that users do not have any user-friendly indicator showing the applications legitimacy, riskiness or it's privacy level. The currently used permission system on Android was shown ineffective by the state of the art studies: users did not understand the meaning of many permissions, was not able to judge the security or legitimacy of permission requests and often ignored the permission lists. We propose «privacy» score and «risk» indicator obtained following an analysis of permission usage of Android applications from different categories. Each application is ranked according to the pattern that defines permission requests presumed normal for a given category. A warning is raised if the «privacy» score go beyond the predefined threshold permitting to detect and inform user about abnormal, abusive or malicious applications before applications installation. The proposed «Privacy» score also permits to compare similar applications regarding they permission request more easily than by comparing permission lists. This contribution enforces «Proactive, not reactive», «End-to-end security», «Visibility and Transparency» and «Respect for the user» principles of «Privacy by Design» in mobile system. The proposed methodology is not only dedicated to final users, but also to mobile application markets owners that should integrate such a system for their users. The behavioral pattern explained in this study was presented on The Seventh International Conferences on Pervasive Patterns and Applications PATTERNS 2015 [133]. The full version of the study is under revision for the Decision Support System Journal, Elsevier.

Third and final contribution of this thesis proposes new permission system model that respects all seven «Privacy by Design» principles. The proposed model is very fine-grained as it is data-oriented. The model integrated vocabulary for building permissions over each piece of data that would clearly show the user what data is used, how and for what purpose. We explain different states of such permissions and also the movement from one state to another. We also propose an example illustrating such system using real applications needing many of private data. Such system embedded into mobile operating systems can become a powerful proactive privacy-respecting and protecting tool and will give mobile systems and mobile users more privacy by design. The study was validated by the research community. Short version of the corresponding article was presented on the The Sixth International Conferences on Pervasive Patterns and Applications PATTERNS 2014 international conference [137] and the extended version was published in International Journal On Advances in Security [138]. We also presented this study during the French Privacy Workshop APVP in 2014.

The full thesis was presented as a poster on PhD forum «What's up Doc!» organized by University of Technology of Troyes in 2015.

8.3 Future work

This thesis proposes patters and models that improves security, visibility, transparency and lack of consent of mobile systems. The future works can be conducted in evaluating

an impact of such pattern integration and developers and also in testing the technology acceptance by users.

This thesis only focused on mobile clients, but the proposed permission system would have an impact on client-server communication and a server side. New research can be conducted on new web service protocols and privacy-friendly APIs. The smartphone should be able to understand which arriving data is sensitive and private so as not to track unrelated data (such as simple news) and to pay additional attention to sensitive data (such as social network communications and profiles). The data tagging could be based on crowd sourcing where users can occasionally verify the incoming data or APIs. The certification and creation of the database of trusted APIs is another option.

The purpose verification system should be proposed as it is a crucial part of the system. The list of goals, such as 'visualize' or 'show personalized advertisements' can be predefined. Language processing techniques can be used to analyze custom purposes and filter for non explicative goals, such as 'improve user experience'. The crowd sourcing user-centric solution may also be tested.

The «Risk» indicator work can be extended by introducing an application recommendation system that could be based on application ratings and privacy scores. Such system could also propose more functional or more privacy-respecting applications in the category according to the defined permission pattern and an application chosen by the user.

The proposed patterns can help mobile systems and developers to respect user's privacy and to be compliant with the european law although they do not force them to do so. A new research can be focused on accountability principle of the Directive: the necessity of the controller to show the compliance of the system with the law. Tools permitting to verify the compliancy and to certify an application could be made as well as tools giving the visibility on the system to verifiers.

Chapter 9

Publications

Peer-review journal articles

- Sokolova, K., Lemercier, M., & Garcia, L. (2014). Towards High Quality Mobile Applications: Android Passive MVC Architecture in International Journal On Advances in Software v7 (1&2), 123 – 138.
- Sokolova, K., Lemercier, M., & Boisseau, J-B. (2014). Respecting user privacy in mobiles: privacy by design permission system for mobile applications in International Journal On Advances in Security v7 (3&4), 110 – 120.

Peer-review International Conference Inproceedings

- Sokolova, K., Lemercier, M., Garcia, L. (2013) Android Passive MVC: a novel architecture model for the Android application development. The Fifth International Conferences on Pervasive Patterns and Applications. IARIA, PATTERNS 2013, Valensia, Spain. Best Paper Award
- Sokolova, K., Lemercier, M., & Boisseau, J-B. (2014). Privacy by Design Permission System for Mobile Applications (pp. 89-95). IARIA, PATTERNS 2014, Vénice, Italie. Best Paper Award
- Sokolova, K., Perez, C., & Lemercier, M. (2015). Android Permissions Usage: a First Step Towards Detecting Abusive Applications. IARIA, PATTERNS 2015, Nice, France.

National presentations without publication and posters

- Sokolova, K., Mobile apps must respect users' privacy (poster). PhD forum «What's up Doc?». University of technology of Troyes. 2015.
- Sokolova, K., Perez, C., & Lemercier, M. Android Permissions Usage: a First Step Towards Detecting Abusive Applications. Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI). University of Technology of Troyes. 19-22 Mai 2015.
- Sokolova, K., Lemercier, M., & Boisseau, J-B. Privacy by Design Permission System for Mobile Applications. 5ème Atelier sur la Protection de la Vie Privée (APVP), Cabourg, 15-18 Juin 2014
- Sokolova, K., & Lemercier, M. Privacy by Design in Mobile Devices. 4ème Atelier sur la Protection de la Vie Privée (APVP), Les Loges en Josas, 17-19 Juin 2013

Chapter 10

Appendix

Tableau 10.1: Permissions extracted from Android 4.4.2

Permission	Level	Group	Description
AUTHENTICATE_ ACCOUNTS	dangerous	ACCOUNTS	Allows the app to use the account authenticator capabilities of the AccountManager
USE_ CREDENTIALS	dangerous	ACCOUNTS	Allows the app to request authentication tokens.
MANAGE_ ACCOUNTS	dangerous	ACCOUNTS	Allows the app to perform operations like adding and removing accounts
CHANGE_ WIFI_ MULTICAST_ STATE	dangerous	AFFECTS_ BATTERY	Allows the app to receive packets sent to all devices on a Wi-Fi network using multicast addresses
GET_ TASKS	dangerous	APP_ INFO	Allows the app to retrieve information about currently and recently running tasks. This may allow the app to discover information about which applications are used on the device.
BLUETOOTH	dangerous	BLUETOOTH_ NETWORK	Allows the app to view the configuration of the Bluetooth on the phone
BLUETOOTH_ ADMIN	dangerous	BLUETOOTH_ NETWORK	Allows the app to configure the local Bluetooth phone
READ_ HISTORY_ BOOKMARKS	dangerous	BOOKMARKS	Allows the app to read the history of all URLs that the Browser has visited
WRITE_ HISTORY_ BOOKMARKS	dangerous	BOOKMARKS	Allows the app to modify the Browser's history or bookmarks stored on your phone. This may allow the app to erase or modify Browser data. Note: this permission may note be enforced by third-party browsers or other applications with web browsing capabilities.

Permission	Level	Group	Description
CAMERA	dangerous	CAMERA	Allows the app to take pictures and videos with the camera. This permission allows the app to use the camera at any time without your confirmation.
SYSTEM_ ALERT_ WINDOW	dangerous	DISPLAY	Allows the app to draw on top of other applications or parts of the user interface. They may interfere with your use of the interface in any application
ACCESS_ FINE_ LOCATION	dangerous	LOCATION	Allows the app to get your precise location using the Global Positioning System (GPS) or network location sources such as cell towers and Wi-Fi. These location services must be turned on and available to your device for the app to use them. Apps may use this to determine where you are
ACCESS_ COARSE_ LOCATION	dangerous	LOCATION	Allows the app to get your approximate location. This location is derived by location services using network location sources such as cell towers and Wi-Fi. These location services must be turned on and available to your device for the app to use them. Apps may use this to determine approximately where you are.
SEND_ SMS	dangerous	MESSAGES	Allows the app to send SMS messages. This may result in unexpected charges. Malicious apps may cost you money by sending messages without your confirmation.
SEND_ SMS	dangerous	MESSAGES	Allows the app to send SMS messages. This may result in unexpected charges. Malicious apps may cost you money by sending messages without your confirmation.
RECEIVE_ SMS	dangerous	MESSAGES	Allows the app to receive and process SMS messages. This means the app could monitor or delete messages sent to your device without showing them to you.
RECEIVE_ MMS	dangerous	MESSAGES	Allows the app to receive and process MMS messages. This means the app could monitor or delete messages sent to your device without showing them to you.
READ_ CELL_ BROADCASTS	dangerous	MESSAGES	Allows the app to read cell broadcast messages received by your device. Cell broadcast alerts are delivered in some locations to warn you of emergency situations. Malicious apps may interfere with the performance or operation of your device when an emergency cell broadcast is received.

Permission	Level	Group	Description
READ_ SMS	dangerous	MESSAGES	Allows the app to read SMS messages stored on your phone or SIM card. This allows the app to read all SMS messages
WRITE_ SMS	dangerous	MESSAGES	Allows the app to write to SMS messages stored on your phone or SIM card. Malicious apps may delete your messages.
RECEIVE_ WAP_ PUSH	dangerous	MESSAGES	Allows the app to receive and process WAP messages. This permission includes the ability to monitor or delete messages sent to you without showing them to you.
RECORD_ AUDIO	dangerous	MICROPHONE	Allows the app to record audio with the microphone. This permission allows the app to record audio at any time without your confirmation.
INTERNET	dangerous	NETWORK	Allows the app to create network sockets and use custom network protocols. The browser and other applications provide means to send data to the internet
CHANGE_ WIFI_ STATE	dangerous	NETWORK	Allows the app to connect to and disconnect from Wi-Fi access points and to make changes to device configuration for Wi-Fi networks.
CHANGE_ WIMAX_ STATE	dangerous	NETWORK	Allows the app to connect the phone to and disconnect the phone from WiMAX networks.
NFC	dangerous	NETWORK	Allows the app to communicate with Near Field Communication (NFC) tags
READ_ PROFILE	dangerous	PERSONAL_ INFO	Allows the app to read personal profile information stored on your device
WRITE_ PROFILE	dangerous	PERSONAL_ INFO	Allows the app to change or add to personal profile information stored on your device
READ_ CALENDAR	dangerous	PERSONAL_ INFO	Allows the app to read all calendar events stored on your phone
WRITE_ CALENDAR	dangerous	PERSONAL_ INFO	Allows the app to add
PROCESS_ OUTGOING_ CALLS	dangerous	PHONE_ CALLS	Allows the app to process outgoing calls and change the number to be dialed. This permission allows the app to monitor
READ_ PHONE_ STATE	dangerous	PHONE_ CALLS	Allows the app to access the phone features of the device. This permission allows the app to determine the phone number and device IDs

Permission	Level	Group	Description
CALL_ PHONE	dangerous	PHONE_ CALLS	Allows the app to call phone numbers without your intervention. This may result in unexpected charges or calls. Note that this doesn't allow the app to call emergency numbers. Malicious apps may cost you money by making calls without your confirmation.
USE_ SIP	dangerous	PHONE_ CALLS	Allows the app to use the SIP service to make/receive Internet calls.
DISABLE_ KEYGUARD	dangerous	SCREENLOCK	Allows the app to disable the keylock and any associated password security. For example
READ_ CONTACTS	dangerous	SOCIAL_ INFO	Allows the app to read data about your contacts stored on your phone
WRITE_ CONTACTS	dangerous	SOCIAL_ INFO	Allows the app to modify the data about your contacts stored on your phone
READ_ CALL_ LOG	dangerous	SOCIAL_ INFO	Allows the app to read your phone's call log
WRITE_ CALL_ LOG	dangerous	SOCIAL_ INFO	Allows the app to modify your phone's call log
READ_ SOCIAL_ STREAM	dangerous	SOCIAL_ INFO	Allows the app to access and sync social updates from you and your friends. Be careful when sharing information – this allows the app to read communications between you and your friends on social networks
WRITE_ SOCIAL_ STREAM	dangerous	SOCIAL_ INFO	Allows the app to display social updates from your friends. Be careful when sharing information – this allows the app to produce messages that may appear to come from a friend. Note: this permission may not be enforced on all social networks.
WRITE_ EXTERNAL_ STORAGE	dangerous	STORAGE	Allows the app to write to the USB storage.
ACCESS_ MOCK_ LOCATION	dangerous	SYSTEM_ TOOLS	Create mock location sources for testing or install a new location provider. This allows the app to override the location and/or status returned by other location sources such as GPS or location providers.
INSTALL_ SHORTCUT	dangerous	SYSTEM_ TOOLS	Allows an application to add Homescreen shortcuts without user intervention.
UNINSTALL_ SHORTCUT	dangerous	SYSTEM_ TOOLS	Allows the application to remove Homescreen shortcuts without user intervention.
SUBSCRIBED_ FEEDS_ WRITE	dangerous	SYSTEM_ TOOLS	Allows the app to modify your currently synced feeds. Malicious apps may change your synced feeds.

Permission	Level	Group	Description
CLEAR_ APP_ CACHE	dangerous	SYSTEM_ TOOLS	Allows the app to free phone storage by deleting files in the cache directories of other applications. This may cause other applications to start up more slowly as they need to re-retrieve their data.
READ_ USER_ DICTIONARY	dangerous	USER_ DICTIONARY	Allows the app to read all words
ADD_ VOICEMAIL	dangerous	VOICEMAIL	Allows the app to add messages to your voicemail inbox.
CHANGE_ CONFIGURATION	development	DEVELOPMENT_ TOOLS	Allows the app to change the current configuration
WRITE_ SECURE_ SETTINGS	development	DEVELOPMENT_ TOOLS	Allows the app to modify the system's secure settings data. Not for use by normal apps.
DUMP	development	DEVELOPMENT_ TOOLS	Allows the app to retrieve internal state of the system. Malicious apps may retrieve a wide variety of private and secure information that they should never normally need.
READ_ LOGS	development	DEVELOPMENT_ TOOLS	Allows the app to read from the system's various log files. This allows it to discover general information about what you are doing with the phone
SET_ DEBUG_ APP	development	DEVELOPMENT_ TOOLS	Allows the app to turn on debugging for another app. Malicious apps may use this to kill other apps.
SET_ PROCESS_ LIMIT	development	DEVELOPMENT_ TOOLS	Allows the app to control the maximum number of processes that will run. Never needed for normal apps.
SET_ ALWAYS_ FINISH	development	DEVELOPMENT_ TOOLS	Allows the app to control whether activities are always finished as soon as they go to the background. Never needed for normal apps.
SIGNAL_ PERSISTENT_ PROCESSES	development	DEVELOPMENT_ TOOLS	Allows the app to request that the supplied signal be sent to all persistent processes.
INTERACT_ ACROSS_ USERS	development	SYSTEM_ TOOLS	Allows the app to perform actions across different users on the device. Malicious apps may use this to violate the protection between users.
SET_ ANIMATION_ SCALE	development	SYSTEM_ TOOLS	Allows the app to change the global animation speed (faster or slower animations) at any time.

Permission	Level	Group	Description
GET_ APP_ OPS_ STATS	development	SYSTEM_ TOOLS	Allows the app to retrieve collected application operation statistics. Not for use by normal apps.
GET_ ACCOUNTS	normal	ACCOUNTS	Allows the app to get the list of accounts known by the phone. This may include any accounts created by applications you have installed.
VIBRATE	normal	AFFECTS_ BATTERY	Allows the app to control the vibrator.
FLASHLIGHT	normal	AFFECTS_ BATTERY	Allows the app to control the flashlight.
WAKE_ LOCK	normal	AFFECTS_ BATTERY	Allows the app to prevent the phone from going to sleep.
TRANSMIT_ IR	normal	AFFECTS_ BATTERY	Allows the app to use the phone's infrared transmitter.
REORDER_ TASKS	normal	APP_ INFO	Allows the app to move tasks to the foreground and background. The app may do this without your input.
RESTART_ PACKAGES	normal	APP_ INFO	Allows the app to end background processes of other apps. This may cause other apps to stop running.
KILL_ BACKGROUND_ PROCESSES	normal	APP_ INFO	Allows the app to end background processes of other apps. This may cause other apps to stop running.
PERSISTENT_ ACTIVITY	normal	APP_ INFO	Allows the app to make parts of itself persistent in memory. This can limit memory available to other apps slowing down the phone.
RECEIVE_ BOOT_ COMPLETED	normal	APP_ INFO	Allows the app to have itself started as soon as the system has finished booting. This can make it take longer to start the phone and allow the app to slow down the overall phone by always running.
MODIFY_ AUDIO_ SETTINGS	normal	AUDIO_ SETTINGS	Allows the app to modify global audio settings such as volume and which speaker is used for output.
SET_ ALARM	normal	DEVICE_ ALARMS	Allows the app to set an alarm in an installed alarm clock app. Some alarm clock apps may not implement this feature.
ACCESS_ NETWORK_ STATE	normal	NETWORK	Allows the app to view information about network connections such as which networks exist and are connected.
ACCESS_ WIFI_ STATE	normal	NETWORK	Allows the app to view information about Wi-Fi networking

Permission	Level	Group	Description
ACCESS_ WIMAX_ STATE	normal	NETWORK	Allows the app to determine whether WiMAX is enabled and information about any WiMAX networks that are connected.
CHANGE_ NETWORK_ STATE	normal	NETWORK	Allows the app to change the state of network connectivity.
EXPAND_ STATUS_ BAR	normal	STATUS_ BAR	Allows the app to expand or collapse the status bar.
READ_ EXTERNAL_ STORAGE	normal	STORAGE	Allows the app to read the contents of your USB storage.
READ_ SYNC_ SETTINGS	normal	SYNC_ SETTINGS	Allows the app to read the sync settings for an account. For example
WRITE_ SYNC_ SETTINGS	normal	SYNC_ SETTINGS	Allows an app to modify the sync settings for an account. For example
READ_ SYNC_ STATS	normal	SYNC_ SETTINGS	Allows an app to read the sync stats for an account
SET_ TIME_ ZONE	normal	SYSTEM_ CLOCK	Allows the app to change the phone's time zone.
ACCESS_ LOCATION_ EXTRA_ COMMANDS	normal	SYSTEM_ TOOLS	Allows the app to access extra location provider commands. This may allow the app to to interfere with the operation of the GPS or other location sources.
WRITE_ SETTINGS	normal	SYSTEM_ TOOLS	Allows the app to modify the system's settings data. Malicious apps may corrupt your system's configuration.
GET_ PACKAGE_ SIZE	normal	SYSTEM_ TOOLS	Allows the app to retrieve its code
BROADCAST_ STICKY	normal	SYSTEM_ TOOLS	Allows the app to send sticky broadcasts
SUBSCRIBED_ FEEDS_ READ	normal	SYSTEM_ TOOLS	Allows the app to get details about the currently synced feeds.
SET_ WALLPAPER	normal	WALLPAPER	Allows the app to set the system wallpaper.
SET_ WALLPAPER_ HINTS	normal	WALLPAPER	Allows the app to set the system wallpaper size hints.
WRITE_ USER_ DICTIONARY	normal	WRITE_ USER_ DICTIONARY	Allows the app to write new words into the user dictionary.
ACCOUNT_ MANAGER	signature	ACCOUNTS	Allows the app to make calls to AccountAuthenticators.
REMOVE_ TASKS	signature	APP_ INFO	Allows the app to remove tasks and kill their apps. Malicious apps may disrupt the behavior of other apps.
MANAGE_ ACTIVITY_ STACKS	signature	APP_ INFO	Allows the app to add
ACCESS_ ALL_ EXTERNAL_ STORAGE	signature	DEVELOPMENT_ TOOLS	Allows the app to access external storage for all users.
HARDWARE_ TEST	signature	HARDWARE_ CONTROLS	Allows the app to control various peripherals for the purpose of hardware testing.

Permission	Level	Group	Description
BROADCAST_ SMS	signature	MESSAGES	Allows the app to broadcast a notification that an SMS message has been received. Malicious apps may use this to forge incoming SMS messages.
BROADCAST_ WAP_ PUSH	signature	MESSAGES	Allows the app to broadcast a notification that a WAP PUSH message has been received. Malicious apps may use this to forge MMS message receipt or to silently replace the content of any webpage with malicious variants.
BLUETOOTH_ STACK	signature	SYSTEM_ TOOLS	-
NET_ ADMIN	signature	SYSTEM_ TOOLS	-
REMOTE_ AUDIO_ PLAYBACK	signature	SYSTEM_ TOOLS	-
INTERACT_ ACROSS_ USERS_ FULL	signature	SYSTEM_ TOOLS	Allows all possible interactions across users.
GET_ DETAILED_ TASKS	signature	SYSTEM_ TOOLS	Allows the app to retrieve detailed information about currently and recently running tasks. Malicious apps may discover private information about other apps.
START_ ANY_ ACTIVITY	signature	SYSTEM_ TOOLS	Allows the app to start any activity
SET_ SCREEN_ COMPATIBILITY	signature	SYSTEM_ TOOLS	Allows the app to control the screen compatibility mode of other applications. Malicious applications may break the behavior of other applications.
FORCE_ STOP_ PACKAGES	signature	SYSTEM_ TOOLS	Allows the app to forcibly stop other apps.
SET_ PREFERRED_ APPLICATIONS	signature	SYSTEM_ TOOLS	Allows the app to modify your preferred apps. Malicious apps may silently change the apps that are run
ASEC_ ACCESS	signature	SYSTEM_ TOOLS	Allows the app to get information on internal storage.
ASEC_ CREATE	signature	SYSTEM_ TOOLS	Allows the app to create internal storage.
ASEC_ DESTROY	signature	SYSTEM_ TOOLS	Allows the app to destroy internal storage.
ASEC_ MOUNT_ UNMOUNT	signature	SYSTEM_ TOOLS	Allows the app to mount/unmount internal storage.
ASEC_ RENAME	signature	SYSTEM_ TOOLS	Allows the app to rename internal storage.
DIAGNOSTIC	signature	SYSTEM_ TOOLS	Allows the app to read and write to any resource owned by the diag group; for example
NET_ TUNNELING	signature	SYSTEM_ TOOLS	-
BROADCAST_ PACKAGE_ REMOVED	signature	SYSTEM_ TOOLS	Allows the app to broadcast a notification that an app package has been removed. Malicious apps may use this to kill any other running app.

Permission	Level	Group	Description
CHANGE_ BACKGROUND_ DATA_ SETTING	signature	SYSTEM_ TOOLS	Allows the app to change the background data usage setting.
GLOBAL_ SEARCH_ CONTROL	signature	SYSTEM_ TOOLS	-
READ_ DREAM_ STATE	signature	SYSTEM_ TOOLS	-
WRITE_ DREAM_ STATE	signature	SYSTEM_ TOOLS	-
STATUS_ BAR_ SERVICE	signature	-	Allows the app to be the status bar.
FORCE_ BACK	signature	-	Allows the app to force any activity that is in the foreground to close and go back. Should never be needed for normal apps.
INTERNAL_ SYSTEM_ WINDOW	signature	-	Allows the app to create windows that are intended to be used by the internal system user interface. Not for use by normal apps.
MANAGE_ APP_ TOKENS	signature	-	Allows the app to create and manage their own tokens
FREEZE_ SCREEN	signature	-	Allows the application to temporarily freeze the screen for a full-screen transition.
INJECT_ EVENTS	signature	-	Allows the app to deliver its own input events (key presses
FILTER_ EVENTS	signature	-	Allows an application to register an input filter which filters the stream of all user events before they are dispatched. Malicious app may control the system UI whtout user intervention.
RETRIEVE_ WINDOW_ INFO	signature	-	Allows an application to retrieve information about the the windows from the window manager. Malicious apps may retrieve information that is intended for internal system usage.
TEMPORARY_ ENABLE_ ACCESSIBILITY	signature	-	Allows an application to temporarily enable accessibility on the device. Malicious apps may enable accessibility without user consent.
MAGNIFY_ DISPLAY	signature	-	Allows an application to magnify the content of a display. Malicious apps may transform the display content in a way that renders the device unusable.
SET_ ACTIVITY_ WATCHER	signature	-	Allows the app to monitor and control how the system launches activities. Malicious apps may completely compromise the system. This permission is only needed for development
GET_ TOP_ ACTIVITY_ INFO	signature	-	Allows the holder to retrieve private information about the current application in the foreground of the screen.

Permission	Level	Group	Description
READ_ INPUT_ STATE	signature	-	Allows the app to watch the keys you press even when interacting with another app (such as typing a password). Should never be needed for normal apps.
BIND_ INPUT_ METHOD	signature	-	Allows the holder to bind to the top-level interface of an input method. Should never be needed for normal apps.
BIND_ ACCESSIBILITY_ SERVICE	signature	-	Allows the holder to bind to the top-level interface of an accessibility service. Should never be needed for normal apps.
BIND_ PRINT_ SERVICE	signature	-	Allows the holder to bind to the top-level interface of a print service. Should never be needed for normal apps.
BIND_ NFC_ SERVICE	signature	-	Allows the holder to bind to applications that are emulating NFC cards. Should never be needed for normal apps.
BIND_ PRINT_ SPOOLER_ SERVICE	signature	-	Allows the holder to bind to the top-level interface of a print spooler service. Should never be needed for normal apps.
BIND_ TEXT_ SERVICE	signature	-	Allows the holder to bind to the top-level interface of a text service(e.g. SpellCheckerService). Should never be needed for normal apps.
BIND_ VPN_ SERVICE	signature	-	Allows the holder to bind to the top-level interface of a Vpn service. Should never be needed for normal apps.
BIND_ REMOTE_ DISPLAY	signature	-	Allows the holder to bind to the top-level interface of a remote display. Should never be needed for normal apps.
BIND_ DEVICE_ ADMIN	signature	-	Allows the holder to send intents to a device administrator. Should never be needed for normal apps.
SET_ ORIENTATION	signature	-	Allows the app to change the rotation of the screen at any time. Should never be needed for normal apps.
SET_ POINTER_ SPEED	signature	-	Allows the app to change the mouse or trackpad pointer speed at any time. Should never be needed for normal apps.
SET_ KEYBOARD_ LAYOUT	signature	-	Allows the app to change the keyboard layout. Should never be needed for normal apps.
CLEAR_ APP_ USER_ DATA	signature	-	Allows the app to clear user data.

Permission	Level	Group	Description
GRANT_ REVOKE_ PERMISSIONS	signature	-	Allows an application to grant or revoke specific permissions for it or other applications. Malicious applications may use this to access features you have not granted them.
ACCESS_ SURFACE_ FLINGER	signature	-	Allows the app to use SurfaceFlinger low-level features.
CONFIGURE_ WIFI_ DISPLAY	signature	-	Allows the app to configure and connect to Wifi displays.
CONTROL_ WIFI_ DISPLAY	signature	-	Allows the app to control low-level features of Wifi displays.
BRICK	signature	-	Allows the app to disable the entire phone permanently. This is very dangerous.
DEVICE_ POWER	signature	-	Allows the app to turn the phone on or off.
FACTORY_ TEST	signature	-	Run as a low-level manufacturer test
CONFIRM_ FULL_ BACKUP	signature	-	Allows the app to launch the full backup confirmation UI. Not to be used by any app.
COPY_ PROTECTED_ DATA	signature	-	copy content
MANAGE_ NETWORK_ POLICY	signature	-	Allows the app to manage network policies and define app-specific rules.
C2D_ MESSAGE	signature	-	-
BIND_ PACKAGE_ VERIFIER	signature	-	Allows the holder to make requests of package verifiers. Should never be needed for normal apps.
ACCESS_ CONTENT_ PROVIDERS_ EXTERNALLY	signature	-	Allows the holder to access content providers from the shell. Should never be needed for normal apps.
ACCESS_ KEYGUARD_ SECURE_ STORAGE	signature	-	Allows an application to access keguard secure storage.
CONTROL_ KEYGUARD	signature	-	Allows an application to control keguard.
BIND_ NOTIFICATION_ LISTENER_ SERVICE	signature	-	Allows the holder to bind to the top-level interface of a notification listener service. Should never be needed for normal apps.
BLUETOOTH_ PRIVILEGED	signatureOrSystem	BLUETOOTH_ NETWORK	Allows the app to pair with remote devices without user interaction.
CAMERA_ DISABLE_ TRANSMIT_ LED	signatureOrSystem	CAMERA	Allows a pre-installed system application to disable the camera use indicator LED.
MANAGE_ USB	signatureOrSystem	HARDWARE_ CONTROLS	Allows the app to manage preferences and permissions for USB devices.
ACCESS_ MTP	signatureOrSystem	HARDWARE_ CONTROLS	Allows access to the kernel MTP driver to implement the MTP USB protocol.
LOCATION_ HARDWARE	signatureOrSystem	LOCATION	-

Permission	Level	Group	Description
SEND_ RESPOND_ VIA_ MESSAGE	signatureOrSystem	MESSAGES	Allows the app to send requests to other messaging apps to handle respond-via-message events for incoming calls.
RECEIVE_ EMERGENCY_ BROADCAST	signatureOrSystem	MESSAGES	Allows the app to receive and process emergency broadcast messages. This permission is only available to system apps.
CONNECTIVITY_ INTERNAL	signatureOrSystem	NETWORK	-
RECEIVE_ DATA_ ACTIVITY_ CHANGE	signatureOrSystem	NETWORK	-
LOOP_ RADIO	signatureOrSystem	NETWORK	-
BIND_ DIRECTORY_ SEARCH	signatureOrSystem	PERSONAL_ INFO	-
RETRIEVE_ WINDOW_ CONTENT	signatureOrSystem	PERSONAL_ INFO	Allows the app to retrieve the content of the active window. Malicious apps may retrieve the entire window content and examine all its text except passwords.
BIND_ APPWIDGET	signatureOrSystem	PERSONAL_ INFO	Allows the app to tell the system which widgets can be used by which app. An app with this permission can give access to personal data to other apps. Not for use by normal apps.
BIND_ KEYGUARD_ APPWIDGET	signatureOrSystem	PERSONAL_ INFO	-
MODIFY_ PHONE_ STATE	signatureOrSystem	PHONE_ CALLS	Allows the app to control the phone features of the device. An app with this permission can switch networks
READ_ PRIVILEGED_ PHONE_ STATE	signatureOrSystem	PHONE_ CALLS	-
BIND_ CALL_ SERVICE	signatureOrSystem	PHONE_ CALLS	Allows the app to control when and how the user sees the in-call screen.
WRITE_ MEDIA_ STORAGE	signatureOrSystem	STORAGE	Allows the app to modify the contents of the internal media storage.
MANAGE_ DOCUMENTS	signatureOrSystem	STORAGE	Allows the app to manage document storage.
MANAGE_ USERS	signatureOrSystem	SYSTEM_ TOOLS	Allows apps to manage users on the device
MOUNT_ UNMOUNT_ FILESYSTEMS	signatureOrSystem	SYSTEM_ TOOLS	Allows the app to mount and unmount filesystems for removable storage.
MOUNT_ FORMAT_ FILESYSTEMS	signatureOrSystem	SYSTEM_ TOOLS	Allows the app to format removable storage.
WRITE_ APN_ SETTINGS	signatureOrSystem	SYSTEM_ TOOLS	Allows the app to change network settings and to intercept and inspect all network traffic

Permission	Level	Group	Description
BATTERY_ STATS	signatureOrSystem	SYSTEM_ TOOLS	Allows an application to read the current low-level battery use data. May allow the application to find out detailed information about which apps you use.
MODIFY_ APPWIDGET_ BIND_ PERMISSIONS	signatureOrSystem	SYSTEM_ TOOLS	-
GLOBAL_ SEARCH	signatureOrSystem	SYSTEM_ TOOLS	-
SET_ WALLPAPER_ COMPONENT	signatureOrSystem	SYSTEM_ TOOLS	-
INSTALL_ LOCATION_ PROVIDER	signatureOrSystem	-	Create mock location sources for testing or install a new location provider. This allows the app to override the location and/or status returned by other location sources such as GPS or location providers.
SET_ TIME	signatureOrSystem	-	Allows the app to change the phone's clock time.
WRITE_ GSERVICES	signatureOrSystem	-	Allows the app to modify the Google services map. Not for use by normal apps.
ALLOW_ ANY_ CODEC_ FOR_ PLAYBACK	signatureOrSystem	-	Allows the app to use any installed media decoder to decode for playback.
MANAGE_ CA_ CERTIFICATES	signatureOrSystem	-	Allows the app to install and uninstall CA certificates as trusted credentials.
STATUS_ BAR	signatureOrSystem	-	Allows the app to disable the status bar or add and remove system icons.
UPDATE_ DEVICE_ STATS	signatureOrSystem	-	Allows the app to modify collected battery statistics. Not for use by normal apps.
UPDATE_ APP_ OPS_ STATS	signatureOrSystem	-	Allows the app to modify collected application operation statistics. Not for use by normal apps.
SHUTDOWN	signatureOrSystem	-	Puts the activity manager into a shutdown state. Does not perform a complete shutdown.
STOP_ APP_ SWITCHES	signatureOrSystem	-	Prevents the user from switching to another app.
BIND_ WALLPAPER	signatureOrSystem	-	Allows the holder to bind to the top-level interface of a wallpaper. Should never be needed for normal apps.
MANAGE_ DEVICE_ ADMINS	signatureOrSystem	-	Allows the holder to add or remove active device administrators. Should never be needed for normal apps.
INSTALL_ PACKAGES	signatureOrSystem	-	Allows the app to install new or updated Android packages. Malicious apps may use this to add new apps with arbitrarily powerful permissions.
DELETE_ CACHE_ FILES	signatureOrSystem	-	Allows the app to delete cache files.

Permission	Level	Group	Description
DELETE_ PACKAGES	signatureOrSystem	-	Allows the app to delete Android packages. Malicious apps may use this to delete important apps.
MOVE_ PACKAGE	signatureOrSystem	-	Allows the app to move app resources from internal to external media and vice versa.
CHANGE_ COMPONENT_ ENABLED_ STATE	signatureOrSystem	-	Allows the app to change whether a component of another app is enabled or not. Malicious apps may use this to disable important phone capabilities. Care must be used with this permission
READ_ FRAME_ BUFFER	signatureOrSystem	-	Allows the app to read the content of the frame buffer.
CAPTURE_ AUDIO_ OUTPUT	signatureOrSystem	-	Allows the app to capture and redirect audio output.
CAPTURE_ AUDIO_ HOTWORD	signatureOrSystem	-	Allows the app to capture audio for Hotword detection. The capture can happen in the background but does not prevent other audio capture (e.g. Camcorder).
CAPTURE_ VIDEO_ OUTPUT	signatureOrSystem	-	Allows the app to capture and redirect video output.
CAPTURE_ SECURE_ VIDEO_ OUTPUT	signatureOrSystem	-	Allows the app to capture and redirect secure video output.
MEDIA_ CONTENT_ CONTROL	signatureOrSystem	-	Allows the app to control media playback and access the media information (title
REBOOT	signatureOrSystem	-	Allows the app to force the phone to reboot.
MASTER_ CLEAR	signatureOrSystem	-	Allows the app to completely reset the system to its factory settings
CALL_ PRIVILEGED	signatureOrSystem	-	Allows the app to call any phone number
PERFORM_ CDMA_ PROVISIONING	signatureOrSystem	-	Allows the app to start CDMA provisioning. Malicious apps may unnecessarily start CDMA provisioning.
CONTROL_ LOCATION_ UPDATES	signatureOrSystem	-	Allows the app to enable/disable location update notifications from the radio. Not for use by normal apps.
ACCESS_ CHECKIN_ PROPERTIES	signatureOrSystem	-	Allows the app read/write access to properties uploaded by the checkin service. Not for use by normal apps.
PACKAGE_ USAGE_ STATS	signatureOrSystem	-	Allows the app to modify collected component usage statistics. Not for use by normal apps.
BACKUP	signatureOrSystem	-	Allows the app to control the system's backup and restore mechanism. Not for use by normal apps.
BIND_ REMOTEVIEWS	signatureOrSystem	-	Allows the holder to bind to the top-level interface of a widget service. Should never be needed for normal apps.

Permission	Level	Group	Description
ACCESS_ CACHE_ FILESYSTEM	signatureOrSystem	-	Allows the app to read and write the cache filesystem.
CRYPT_ KEEPER	signatureOrSystem	-	-
READ_ NETWORK_ USAGE_ HISTORY	signatureOrSystem	-	Allows the app to read historical network usage for specific networks and apps.
MODIFY_ NETWORK_ ACCOUNTING	signatureOrSystem	-	Allows the app to modify how network usage is accounted against apps. Not for use by normal apps.
MARK_ NETWORK_ SOCKET	signatureOrSystem	-	Allows the app to modify socket marks for routing
PACKAGE_ VERIFICATION_ AGENT	signatureOrSystem	-	Allows the app to verify a package is installable.
SERIAL_ PORT	signatureOrSystem	-	Allows the holder to access serial ports using the SerialManager API.
UPDATE_ LOCK	signatureOrSystem	-	Allows the holder to offer information to the system about when would be a good time for a noninteractive reboot to upgrade the device.
ACCESS_ NOTIFICATIONS	signatureOrSystem	-	Allows the app to retrieve
INVOKE_ CARRIER_ SETUP	signatureOrSystem	-	Allows the holder to invoke the carrier-provided configuration app. Should never be needed for normal apps.
ACCESS_ NETWORK_ CONDITIONS	signatureOrSystem	-	Allows an application to listen for observations on network conditions. Should never be needed for normal apps.

Chapitre 11

Résumé

11.1 Contexte

Le premier téléphone mobile a été inventé en 1908 et depuis, il a beaucoup évolué. Dorénavant, les smartphones et les tablettes reçoivent, stockent et transfèrent une large quantité de données en proposant des services répondant à tous les besoins des utilisateurs à l'aide d'applications mobiles facilement téléchargeables et installables. Un grand nombre des capteurs intégrés dans un smartphone permet à l'appareil de récolter de l'information très précise sur l'utilisateur et son environnement à tout moment.

Cette importante quantité de données privées comme professionnelles devient difficile à gérer. De plus, elles attirent les acteurs malveillants. Les leaders du marché, Apple et Google ont des stratégies différentes concernant leurs services. Les applications Apple disponibles sur le kiosque de distribution des applications officielles ne contiennent que les applications vérifiées et certifiées par l'entreprise. Google propose seulement une vérification automatique détectant les empreints de la malveillance connus sans pour autant certifier les applications. De telle manière, Google Play - le kiosque de distribution des applications proposé par Google - contient des applications qui peuvent collecter et utiliser des données des utilisateurs d'une manière abusive voire à des fins malveillantes. Les applications contiennent aussi les failles de sécurité. Les utilisateurs mobiles se sentent concernés car, auparavant, jamais les données privées n'étaient générées, collectées et utilisées si massivement qu'actuellement avec les smartphones et les réseaux sociaux [1].

Dr. Ann Cavoukian a observé la tendance d'utilisation des données privées et l'approche réactive de la sécurité en proposant les mises à jour rapides. En 2001, elle a proposé l'approche « Privacy by Design » qui se résume à sept principes intégrant la notion de respect des données privées dans les systèmes de phase de conception [2]. Ces principes ne décrivent pas des solutions concrètes technologiques, mais expliquent les concepts qui doivent être adaptés et appliqués à chaque technologie. L'adoption de ces principes est assez limitée pour le moment, surtout en considérant que l'information créer de la valeur et se monétise. La question se pose : « Pourquoi les développeurs des applications mobiles vont appliquer ces concepts si cela n'est pas une obligation ? »

En Europe, la directive européenne de la protection des données privées (Directive 95/46/EC) a été adoptée en 1995 et elle intègre les principes du « Privacy by Design ». La nouvelle loi européenne unifiée (General Data Protection Regulation) renforce la protection et le respect des données privées en prenant en compte des technologies modernes et rend « Privacy by Design » une obligation légale dans l'Union Européenne.

11.2 Problématique

Les systèmes d'information ne prennent pas toujours en compte les idées du « Privacy by Design », principalement à cause d'un manque de compréhension des ces notions et aussi d'un manque des patrons de conception et de développement liée à ces notions. De nombreux rapports tels que [7] répètent des principes et des recommandations généraux concernant le respect des données privées mais les solutions techniques sont guère évoqués ou expliqués.

Concernant les systèmes mobiles, « Opinion 02/2013 on apps on smart devices' by the Article 29 Data Protection Working Party » [8] a été adopté en 2013 et indique clairement que la régulation européenne est applicable aux systèmes et aux applications mobiles. L'article souligne quatre problèmes principaux liés aux systèmes mobiles et au respect des données privées : manque de transparence d'utilisation des données, absence de consentement, sécurité faible et mépris de but d'utilisation des données.

Le but de cette thèse est de proposer des solutions pour améliorer les quatre points évoqués dans [8] et de rendre les applications et les systèmes mobiles plus « Privacy by (re)Design ».

Cette thèse s'appuie sur la régulation européenne car celle-ci est la plus récente et la plus sévère, comparée aux régulations sur le respect des données privées existant au Canada ou aux Etats-Unis. La thèse propose seulement des solutions mobiles client. Les modèles liés au serveur et à la communication client-serveur sont hors-champ du travail mené.

Le système Android a été choisi pour les études de cas pour de multiples raisons. Tout d'abord, Android est l'un des leaders du marché. Puis, le système est ouvert et accessible en proposant plus de liberté et de l'information pour la recherche, mais attire en même temps des acteurs malveillants. Finalement, les travaux de l'état de l'art ont montré de nombreuses limites de ces systèmes. Notons que, même si les travaux de cette thèse ont été menés sur Android, les solutions proposées sont génériques et peuvent être appliquées aux autres systèmes.

11.3 Questions de recherche

La thèse se focalise sur quatre problématiques définies dans [8] et dont le but est de répondre à la question de recherche suivante : « Pouvons-nous proposer des patrons de conceptions visant à améliorer la transparence, la sécurité, le consentement qui aideront aux applications et aux systèmes mobiles à mieux respecter la vie privée des utilisateurs ». La thèse se concentre sur deux axes : l'architecture des applications mobiles et les systèmes de permissions mobiles. Les limites suivantes ont été identifiées dans l'état de l'art :

1. La qualité du code des applications Android est médiocre : manque d'une architecture unifiée

2. Un système de permission critiqué et un manque d'indicateur de risque compréhensible par les utilisateurs

En se basant sur ces limites, la thèse répond aux questions suivantes :

- Quel modèle d'architecture sera le plus adapté au développement des applications Android ?
- Pouvons-nous générer un indicateur de respect de la vie privée en utilisant de l'information liée aux applications telles que les permissions requises ?
- Pouvons-nous améliorer les systèmes de permission existants pour les rendre plus « Privacy by Design » ?

11.4 Vue sur le respect de la vie privée et « Privacy by Design »

Le droit à la vie privée est protégé dans de nombreux pays comme, par exemple, les Etats-Unis (« Children's Online Privacy Protection Act » (COPPA) [3] et « California Online Privacy Protection Act of 2003 » (OPPA) [4] et le Canada (Personal Information Protection and Electronic Documents Act (PIPEDA) [5]). En Union European, la protection de la vié privée faite parti des droits fondamentaux de l'homme est protégé par la Directive 95/46/EC [9] adopté en 1995. La Commission Européenne a dévoilé un projet de règlement sur la protection des données européennes qui fait appliquer les mêmes principes dans toute l'Union européenne [11]. La Directive sera donc remplacée par « European Data Protection Regulation » [10] pour homogénéiser et renforcer la protection de la vie privée dans l'Union Européenne sans avoir besoin des implémentations locales des lois de chaque pays. L'application de la réglementation doit être mise en place en en décembre 2017.

Une donnée personnelle est définie par la Directive comme une donnée liée à une personne identifiable directement ou indirectement. N'importe quel traitement des données privées doit être fait en respectant les principes décrits dans la Directive. Une donnée personnelle liée à la santé, l'opinion politique, la vie sexuelle et d'autres données à risque est définie comme étant sensible et nécessitant une protection supplémentaire. La Directive définit cinq principes liés au traitement des données privées qui doivent être respectées par un système : traitement licite ; spécification et limitation du but de traitement ; qualité des données ; traitement équitable ; responsabilité.

- Le **traitement licite** signifie que le traitement poursuit un but légitime et conforme à la loi.
- Le principe de **spécification et limitation du but de traitement** signifie que le but de traitement doit être clair et visible avant que le traitement ne soit effectué. Les données personnelles ne peuvent pas être utilisées au-delà du but défini.
- Le principe de la **qualité des données** signifie que les données doivent être adéquates, pertinentes et non excessives au regard but défini ; les données doivent être exactes et rendues anonymes ou supprimées dès que l'objectif est atteint.

- **Traitement équitable** signifie qu'une personne concernée doit pouvoir clairement comprendre laquelle de ses données privées est utilisée, comment, qui utilise ces données et à quelles fins. Le système devrait demander aux utilisateurs un consentement valable (clair et sous aucune pression). Enfin, l'utilisateur doit avoir un libre accès aux données recueillies.
- Finalement, le principe de la **responsabilité** signifie que l'organisme doit assurer la sécurité des données personnelles et doit être en mesure de démontrer la conformité du traitement de données avec les principes de protection des données.

La directive inclut d'une manière explicite ou implicite les sept principes du « Privacy by design » :

- **Proactive, et non réactive**

La vie privée doit être une partie importante de la conception du système : des possibilités d'abus des données privées doivent être identifiées au cours de la conception du système.

- **Protection de la vie privée considérée comme un paramètre par défaut**

Tous les systèmes doivent être conçus en utilisant les principes de protection des données privées tels que la minimisation des données, la non-traçabilité, l'agrégation de données, la limitation de l'utilisation et de la spécification de but d'utilisation.

- **Respect de la vie privée intégré dans les systèmes de la conception**

Les caractéristiques spécifiques au système lié à la protection des données privées doivent être identifiées et intégrées de la conception par exemple à l'aide des patrons de conceptions connus.

- **La fonctionnalité complète**

Ni les fonctionnalités du système, ni la sécurité, ni le respect de la vie privée des utilisateurs ne doivent pas être sacrifiés.

- **La sécurité des données à travers le cycle de vie complet des données**

Les mécanismes de sécurité appropriés doivent être appliqués pour assurer la sécurité des données à travers le cycle de vie complet des données.

- **Visibilité et transparence**

L'utilisateur doit savoir et comprendre quelles données sont utilisées à quelles fins et doit être en mesure de contrôler ses données privées. Les paramètres par défaut du système doivent respecter la vie privée et l'utilisateur doit les changer lui-même pour être plus exposé.

- **Respect de l'utilisateur**

Par défaut, la confidentialité de l'utilisateur doit être protégée sans aucune action supplémentaire de l'utilisateur.

Cette thèse analyse les systèmes mobiles vis-à-vis du concept « Privacy by Design » et propose un certain nombre de modèles de sécurité et du respect de la vie privée pour les systèmes mobiles couvrant l'ensemble des sept principes du « Privacy by Design ».

11.5 Contribution

Cette section présente brièvement les travaux réalisés durant cette thèse. La première sous-section décrit « Android Passive MVC » - une architecture adaptée dédiée aux développeurs Android. La deuxième sous-section présente la méthodologie de mesure du respect des données privées ainsi que la méthodologie de détections des applications anormales et abusives. La troisième sous-section présente un nouveau système de permission mobile qui respecte les notions du « Privacy by Design ».

11.5.1 Android Passive MVC

De nos jours, la demande de développement mobile est très élevée. Pour être compétitive, une application mobile doit être rentable et de bonne qualité. Un code qui est mal écrit peut non seulement avoir de bogues et être difficile à maintenir, mais il peut également devenir une source importante de failles de sécurité. Les entreprises, telles que EUTECH SSII, qui développe les applications iOS et Android pour ses clients, ont noté que les développeurs d'applications Android ont plus de difficultés que les développeurs iOS, non seulement avec des questions de sécurité, mais aussi avec l'architecture et l'organisation du code. De ce fait, les applications Android, non seulement plus coûteuses et longues à développer, mais aussi complexes à mettre à jour et à maintenir, surtout par les développeurs différents.

Le choix de modèle de conception architecturale est important pour veiller sur la qualité de l'application, pour assurer la maintenabilité, pour simplifier l'évolution et les mises à jour, mais aussi pour réduire le temps de développement. Deux entreprises principales sont largement représentées sur le marché du mobile : Apple (iOS) et Google (Android). Le développement iOS est basé sur le modèle de conception Modèle-Vue-Contrôleur adapté et bien structuré.

L'architecture Modèle-Vue-Contrôleur a été proposée en 1978 et depuis, elle est largement utilisée et appliquée aux nombreux langages de programmation [19, 20, 21]. L'objectif de ce modèle est de séparer la logique métier de la logique de présentation ; les modifications de la logique métier ne doivent pas affecter la logique de présentation et vice versa [19]. MVC se compose de trois éléments principaux : Modèle, Vue et Contrôleur. Le Modèle représente les données à afficher sur l'écran. Plus généralement, le Modèle est un Modèle de domaine qui contient la logique métier, les données à manipuler et des objets d'accès aux données. La Vue est une composante visuelle sur l'écran, comme un bouton. Le Contrôleur gère les événements liés aux actions de l'utilisateur et communique avec le modèle. Deux types du MVC existent : MVC Classique et MVC Passif. Dans MVC Classique, la communication entre la Vue et le Model est effectuée à l'aide du patron de conception nommé Observateur-Observable. Dans MVC Passif, la communication entre la Vue et le Model est effectuée via le Contrôleur.

L'architecture de base pour les applications mobiles iOS est un MVC Passif adapté. Comme le MVC Passif original, l'architecture iOS est basée sur trois composants : Vue, Modèle et Contrôleur. Les Modèles et les Vues sont indépendants et communiquent entre eux uniquement par les Contrôleurs. La communication entre les Contrôleurs et le Modèle est organisée via le patron de conception Observateur-Observable. Un nombre des Vues et des Contrôleurs réutilisables est déjà disponible pour les développeurs iOS.

Google n'impose aucune architecture particulière aux développeurs, mais propose des composants différents pour des besoins particuliers. Une description exhaustive de l'environnement et des modules de développement Android peut être trouvée dans [29]. Les développeurs Android ont quatre principales composantes à leur disposition : activité, service, fournisseur de contenu et récepteur de diffusion. Les développeurs utilisent les classes extensibles prédéfinies pour mettre en œuvre ces composants. Une activité est une composante principale et incontournable d'une application Android créée lorsque l'application est ouverte. Une application Android simple peut contenir seulement une activité. L'activité est également le point d'entrée dans l'application : pour démarrer une application, le système doit lancer une activité.

Le système Android ne nécessite pas de suivre une architecture définie : le choix de l'architecture et la qualité de l'application dépendent fortement de l'expérience de développeur. Cette tâche peut être particulièrement difficile pour les développeurs moins expérimentés. Les applications complexes qui ne possèdent pas d'architecture peuvent finir comme une «grosse boule de boue» : incompréhensible et difficile à maintenir [108]. De plus, le code complexe pourrait non seulement conduire à des bogues car il est difficile à tester, mais peut aussi affecter la sécurité d'une application [109]. Les solutions hétérogènes ralentissent le développeur, tandis que le modèle de conception uni et connu pourrait non seulement améliorer le temps de développement, mais aussi améliorer la maintenabilité, l'extensibilité et la performance de l'application.

Les livres et les tutoriels de développement Android sont principalement concentrés sur les détails techniques du SDK Android et sur la conception de l'interface utilisateur. Seuls quelques travaux ont été consacrés à l'architecture de l'application Android, tandis que la communauté Android identifie une architecture comme étant une partie importante de la conception du système et le développement réussis. Les développeurs ouvrent de nombreuses discussions sur l'architecture appropriée pour les applications Android sur les forums, les blogs et les groupes.

Cette thèse étudie les modèles de conception architecturale existantes et propose un modèle d'architecture unifiée et adaptée au développement Android : Android Passif MVC. Nous avons d'abord analysé les patrons de conception architecturaux existants tels que Model-Vue-Contrôleur, Model-Vue-Présenter et Présentation-Abstraction-Contrôle. Puis, Nous avons étudié les choix d'architecture faits par des développeurs Android. Nous avons intégré l'équipe de développement Android d'EUTECH SSII pour observer des solutions utilisées pour les applications existantes. Nous avons interrogé les développeurs sur les choix d'architecture et leurs utilisations dans le processus de développement Android. Nous avons également effectué des recherches à propos des questions liées à l'architecture Android sur StackOverflow - la plus grande communauté de développeurs de logiciels répondant aux questions relatives aux logiciels - et blogs pour lister un ensemble de solutions et leurs inconvénients.

Après cette phase, Nous avons conclu que la difficulté principale des développeurs Android réside dans le choix des rôles des composantes natives comme Activité et Fragment dans une architecture telle que Model-Vue-Contrôleur. L'Activité est une composante principale et incontournable sur Android qui est difficilement réutilisable. Le Fragment est une composante réutilisable apparue dans le but de simplifier l'adaptabilité de l'interface utilisateur aux appareils de tailles différents. Certaines Vues sont déjà intégrées dans le SDK Android et les Vues sont souvent combinées à l'aide d'un code XML. Les Activités et les Fragments sont aussi attachés aux Vues via un processus prévu par le sys-

tème. Nous avons observé, que l'Activité qui joue le rôle d'un Contrôleur devient très vite complexe du fait que la complexité de l'interface augmente ; cependant, les Fragments aident à découper l'application et son interface en modules permettant de simplifier le code. Après un nombre de modèles et prototypes créés, évalués et testés au sein de l'entreprise EUTECH SSII, nous proposons donc une architecture hiérarchique qui se base sur Model-Vue-Contrôleur construit autour du composant Activité.

- Une Vue représente une partie de l'interface.
- Un Contrôleur intercepte des événements d'une Vue.
- Le Model représente des données et, plus généralement, le cœur de l'application.

L'Activité joue le rôle d'un Contrôleur principal ou d'un Contrôleur parent qui gère l'écran entier et l'échange entre les Contrôleurs fils. Les triades MVC sont construites autour de l'Activité et les Fragments peuvent jouer le rôle des Contrôleurs. La communication entre les Vues et le Model est fait à travers le Contrôleur. La communication entre les Fragments, si c'est nécessaire, se doit d'être faite à l'aide du Contrôleur parent ou de l'Activité pour rendre les composantes indépendantes. Nous intégrons deux types de Contrôleurs dans l'architecture qui se diffèrent par leurs rôles : le contrôleur de coordination et le contrôleur médiat. Le contrôleur de coordination est un Contrôleur classique qui correspond à un événement utilisateur issu d'une action faite sur la partie de l'interface dédiée. Le contrôleur médiat gère les autres contrôleurs et permet, par exemple, d'échanger une partie de l'interface avec une autre ou d'échanger les activités. Un menu est un exemple du contrôleur médiat. Une activité peut jouer le rôle du contrôleur médiat dans une application simple.

Android Passif MVC rend l'activité beaucoup moins complexe en déplaçant toute la gestion d'événements, la logique de présentation dans le contrôleur et la gestion de l'interface dans les Vues. Les Vues et les Contrôleurs créés à la demande aident à éviter la création des objets inutiles en économisant la mémoire. Les Vues Android prédéfinies correspondent à l'architecturé proposée et de nouvelles Vues qui pourraient être créées par le développeur seront réutilisables dans des applications futures. Les contrôleurs de coordination sont réutilisables et rendent l'application très modulaire. Les contrôleurs médiats sont moins réutilisables, mais permettent de modifier facilement la logique de l'application qu'en modifiant ses contrôleurs.

Les développeurs peuvent facilement modifier ou supprimer des composants de l'application en modifiant ou supprimant la couple Vue-Contrôleur correspondant. L'application peut être étendue avec des nouvelles couples Vue-Contrôleur. Le modèle est indépendant de la vue, le contrôleur et l'activité. L'interface utilisateur peut être remplacée sans aucun impact sur le modèle, ce qui rend l'application plus maintenable.

Nous avons évalué l'architecture finale vis-à-vis des critères de la qualité du code par scénarios et par des développeurs Android. Après de multiples tests, Nous avons établi un prototype d'illustration de l'architecture et fixé plusieurs scénarios d'évolution du prototype avec les modifications diverses de l'interface. Le prototype s'est montré résistant aux changements en montrant que l'architecture facilité la maintenance, extensibilité et réutilisabilité des composants dans une application. Nous avons aussi demandé à un développeur Android d'EUTECH SSII de redévelopper l'un de ses projets complexes et non

maintenables en utilisant notre architecture. Nous avons pris des métriques diverses du code telles que le nombre des méthodes, le nombre des classes et la complexité du code. Toutes les métriques ont été considérablement réduites dans la version de l'application implémentée avec Android Passive MVC. Une évaluation longue au sein d'EUTECH SSII a montré que l'architecture a permis de simplifier le travail des développeurs Android et d'améliorer la qualité des applications produites par les développeurs. Ces derniers ont noté que l'architecture a facilité le travail en équipe en proposant un cadre connu par tous les membres, mais a également simplifié le travail des stagiaires.

L'architecture proposée, même si elle a été adaptée pour Android, n'est pas dépendante du système Android. Toutes les composantes peuvent être développées en utilisant les classes simples sans activité ou fragment. L'architecture rejoint le groupe des architectures comme Model-Vue-Contrôleur, Model-Vue-Présenter et Présentation-Abstraction-Contrôle. Elle reste une architecture générale.

11.5.2 Indicateur du respect de la vie privée

Android est l'un des leaders du marché mobile, offrant plus d'un million d'applications sur Google Play Store. Google vérifie les applications vis-à-vis des empreintes malveillants connus. Cependant, les applications qui collectent abusivement les données des utilisateurs et demandent les accès à des services sensibles non liés à des fonctionnalités sont toujours présentes sur le marché.

Les informations sur chaque application disponible sur les kiosques des applications mobiles aident les utilisateurs à choisir l'application la plus appropriée ; elles sont similaires pour les différents kiosques. Ces informations comprennent le nom, la description, la note, les commentaires, les captures d'écran, l'icône et, parfois, la politique de confidentialité. Le kiosque des applications Android GooglePlay comprend aussi la liste des accès requis pour certaines interfaces ou données sensibles nommées permissions. Un système de permissions est une solution de sécurité centrée sur l'utilisateur contre les applications et les logiciels malveillants ou abusifs. Cette liste des permissions est censée avertir les utilisateurs sur les applications dangereuses et abusives.

L'état de l'art a montré que les systèmes de permissions actuelles ne sont pas adaptés pour les utilisateurs finaux : les utilisateurs sont incapables de comprendre et de juger les permissions requises par chaque application et souvent ignorent les avertissements lors de l'installation. Tout d'abord, la liste des permissions est présentée uniquement à la dernière étape avant l'installation de l'application lorsque les autres critères de la décision de l'utilisateur ont été atteints, et, par conséquent, la liste des permissions est considérée comme une obligation d'être vu plutôt qu'un facteur de décision [46, 45]. Souvent, les utilisateurs ne disposent pas de suffisamment d'information pour comprendre le sens des permissions, leur but d'utilisation et le mal possible si la permissions est donnée. Certaines permissions sont requises si fréquemment que les utilisateurs ne font aucune attention à celles-ci [39, 40]. On peut voir qu'actuellement, il n'y a pas de système efficace et simple pour les utilisateurs qui leur permette de choisir une application en prenant en compte le niveau du respect de sa vie privée et la sécurité de l'application Android. Les utilisateurs doivent, soit compter sur la communauté avec des commentaires et des évaluations (qui se réfèrent rarement à des problèmes de sécurité), soit vérifier manuellement les autorisations et compter sur leurs connaissances et leur compréhension personnelle.Les travaux de l'état

de l'art suggèrent l'intégration de nouveaux indicateurs de sécurité et de confidentialité pour les utilisateurs [89].

Toutefois, la liste des permissions Android fournit des informations sur le comportement de l'application et peut être appropriée pour l'analyse automatique des applications. Android inclut environ 80 permissions disponibles aux développeurs tiers et encore le double des permissions dédiées aux applications des fournisseurs des appareils mobiles. Chaque nouvelle version Android contient de plus en plus de permissions liées aux nouvelles fonctionnalités et aux nouveaux capteurs. L'information sur les permissions requises est intégrée dans chaque application et elle est toujours disponible. L'identification des permissions-clés pour des fonctionnalités différentes ainsi que des demandes de permissions attendues peut aider à levier le comportement anormal de l'application et de fournir un indicateur du risque simple qui peut avertir les utilisateurs. Les applications avec des fonctionnalités similaires sont regroupées sur Google Play en différentes catégories. Cette thèse analyse donc les demandes de permissions par catégorie. Cette étude propose une méthodologie pour caractériser un comportement normal pour les applications de chaque catégorie, en soulignant les demandes de permissions attendues. Cette étude vise à répondre aux questions de recherche suivantes :

- Est-ce que les applications Android de différentes catégories nécessitent des différents groupes de permissions et est-ce que ces groupes des permissions représentent ainsi la catégorie ?
- Est-ce qu'un groupe de permissions-clés par catégorie peut nous permettre de mesurer le niveau de risque et le niveau de respect de la vie privée de l'utilisateur d'une application ?
- Peut-on détecter les applications malveillantes et abusives en terme de permissions à l'aide des groupes de permissions-clés des catégories ?

Pour réaliser cette étude, une collecte des applications mobiles a été effectuée sur le kiosque officiel des applications Android - GooglePlay. Le crawler que nous avons réalisé a pu obtenir 9.512 applications des 35 catégories différentes. Pour chaque application, on a extrait un nom, une description et une liste de permissions associées. On a ensuite filtré les listes de permissions afin de supprimer toutes les permissions erronées ou créées par des développeurs pour ne garder que les permissions officielles intégrées dans Android 4.4.

Les données obtenues ont été utilisées pour extraire des patterns de permissions pour chaque catégorie d'applications. Pour chaque catégorie, nous avons modélisé des permissions sous forme de graphes. Deux permissions qui sont requises conjointement par une application de la catégorie visée sont liées dans le graphe. On a calculé le pois les lient des graphes obtenu en utilisant le Z-score. Il est connu que Google Play peut contenir des applications de mauvaise qualité et même malveillantes ; à ce titre, leur présence dans notre base de données n'a rien de surprenant. Pour éviter les permissions anormales et négligeables dans notre modèle, la troisième étape omet des paires de permissions avec une occurrence moyenne inférieure à 1.

Le graphe est ensuite filtré en se basant sur le Z-score pour obtenir que la permission pertinente pour la catégorie par rapport à l'utilisation du couple des permissions dans les autres catégories. Par définition, le Z-score sera négatif si la fréquence observée d'une paire

de permissions dans une catégorie est inférieure à la moyenne. Le Z-score sera égal à 1 si la fréquence observée est supérieure à la moyenne exactement par la mesure de l'écart type. Enfin, le Z-score sera supérieur à 1 si la fréquence d'une paire de permissions donnée est nettement supérieure à la moyenne. Pour obtenir le graphe final pour chaque catégorie, on a filtré les paires de permissions pour ne garder que les couples dont le Z-score est supérieur à 1. Les nœuds non connectés sont également omis.

Les permissions sont ensuite analysées avec les méthodes d'analyse de graphe. Les métriques obtenues révèlent les permissions centrales pour chaque catégorie. Pour chaque nœud, on a calculé les mesures telles que le degré, le degré pondéré, la centralité d'intermédiarité, la centralité de proximité, le Page Rank, le Hub et l'Autorité. De telle manière, on a obtenu un motif de permissions pour chaque catégorie qui contient une liste de permissions les plus représentatives pour une catégorie donnée.

Pour évaluer la représentativité d'un motif par rapport à une catégorie, on a vérifié la performance de la classification des applications en catégorie en utilisant des motifs obtenus. Pour effectuer la classification, on a proposé un attribut de similarités d'une application par rapport à un motif d'une catégorie donnée et d'une mesure donnée. On a ensuite évalué la classification par la validation croisée en utilisant les mesures obtenues précédemment. La classification avec Naive Bayesien s'est montrée la plus performante, ce qui a montré que les catégories sont bel et bien représentées par des motifs de permissions centrales. Deux métriques se sont révélé comme étant les plus performantes : centralité d'intermédiarité et degré pondéré. Certaines catégories étaient plus difficiles à prédire : par exemple, certaines applications de la catégorie 'transport' ont été classifiées comme appartenant a la catégorie 'voyage'.

Nous avons aussi comparé les motifs obtenus avec l'indicateur le plus présent dans l'état de l'art – l'occurrence d'une permission. Dans notre base de données, les permissions les plus utilisées dans chaque catégorie se sont révélées identiques, ainsi que les motifs obtenus (par rapport à la centralité d'intermédiarité) permette de comprendre les catégories d'applications et leurs fonctionnalités ainsi que mieux décrire la catégorie.

En se basant sur les motifs obtenus par catégorie et sur les deux métriques les plus performantes, on a ensuite proposé un indicateur pour évaluer le niveau de respect des données privées par l'application. L'indicateur montre à quel point l'application suit le motif précedemment trouvé par rapport à sa catégorie. On a construit l'indicateur en se basant sur les prérequis suivantes :

- Plus l'application tend vers le motif, plus la valeur est élevée.
- Si l'application requiert des permissions qui sont centrales dans le motif de sa catégorie, plus la valeur doit être élevée.
- Si l'application demande des permissions qui ne sont pas présentes dans le motif, la valeur de l'indicateur doit diminuer.

Enfin, cette étude évalue la performance de détection des applications malveillantes par rapport à l'indicateur et le score du respect de la vie privée défini précédemment. Pour cela, on prend un ensemble d'applications non malveillantes d'une catégorie et on ajoute un ensemble d'applications connues comme étant malveillantes. On a effectué la classification des applications dans deux catégories (malveillant et non-malveillant) en

modifiant des variables d'ajustement de l'indicateur du respect de la vie privée. Ainsi, on a obtenu un seuil à partir duquel l'application est considérée comme étant une application à risque et une alerte est levée pour informer l'utilisateur.

11.5.3 Système de permissions « Privacy by Design »

Un des mécanismes de sécurité des données mobiles est le mécanisme de contrôle d'accès appelé système de permissions. Un système de permissions est intégré dans les systèmes d'exploitation mobiles et il est crucial pour la protection et le respect des données privées des utilisateurs mobiles. Si le modèle d'un tel système est bien construit, un système de permission peut devenir un outil de protection de la vie privée puissant et proactif faisant partie du système d'exploitation.

Les nombreux travaux de l'état de l'art montrent que les systèmes de permission actuels n'assurent ni le respect de la vie privée ni la sécurité [39, 40, 38, 42]. Les utilisateurs ne peuvent ni juger le risque d'une permission ni le comprendre. Finalement, les alertes liées aux permissions utilisées par l'application sont souvent ignorées par les utilisateurs. Un nouveau système de permission peut donner aux utilisateurs non seulement le contrôle sur leurs données, mais aussi la transparence et la visibilité sur l'exploitation de leurs données.

Au lieu de définir une liste de permissions fix comme cela est implémenté dans les systèmes mobiles actuels, nous proposons dans cette thèse un modèle eu le vocabulaire pour la création et l'utilisation des permissions. On se base sur un contrôle d'accès discrétionnaire où le Sujet a un certain Droit sur un Objet donné par le propriétaire des données – utilisateur.

- Le Sujet, dans notre contexte, est une application mobile représentée par son nom ou identifiant unique.
- Un Objet est une donnée privée d'une certaine granularité : numéro de téléphone, nom, prénom, mais aussi les photos, les médias, les listes d'amis, les données de géolocalisation, les données de la santé, etc.
- Le Droit dans notre modèle est une combinaison de l'Action et du But d'utilisation.

L'Action représente les traitements différents qui peuvent être faits sur la donnée : télécharger, sauvegarder, modifier, accéder ou transmettre. Le But est défini par le développeur et doit être précis. Les exemples du But bien défini sont les suivants :

- Afficher sur l'écran
- Calculer et afficher vos performances sportives
- Contacter vos amis
- Avoir les publicités personnalisées
- Calculer un score du risque
- Publier sur le mur du Facebook

Le But ne doit par être vague, comme, par exemple, les buts suivants qui sont souvent utilisés par les applications et les systèmes mobiles :

- Améliorer l'expérience utilisateur
- Pour faire fonctionner l'application correctement

Pour une permission liée à la sauvegarde de nouvelles données, le développeur doit aussi définir le temps de stockage ou proposer à l'utilisateur de limiter le temps de stockage par le nombre d'heures et le nombre de jours. La limitation peut aussi se baser sur le cycle de vie de l'application en proposant des explications telles que :

- Jusqu'à la suppression de l'application
- Jusqu'à la fermeture de l'application
- Jamais

L'Utilisateur doit être notifié à propos de la permission et doit l'autoriser explicitement. L'utilisateur peut choisir d'autoriser ou de révoquer d'une manière permanente la permission. L'utilisateur peut dire s'il souhaite ou non avoir une confirmation à chaque fois qu'il utilise cette permission. Pour les autorisations sans confirmation, l'utilisateur doit pourvoir changer le statut de chaque permission dans les paramètres de l'application. Les permissions peuvent être utilisées par l'application seulement en respectant les conditions données par l'utilisateur. Une permission peut être utilisée suite à une action utilisateur (par exemple, « appui sur un bouton précis ») ou à un évènement (par exemple, « lancement de l'application » ou « envoi du message »). Une action automatique liée à un événement de l'application peut être aussi restreinte par une fréquence d'utilisation (par exemple « une fois par jour »).

Les permissions qui ont le même But d'utilisation peuvent être groupées ensemble. Le groupe peut avoir un statut « tout » ou « un ». Si le statut du groupe est « tout », cela signifie que toutes les permissions sont nécessaires pour assurer la fonctionnalité définie par le But. Dans ce cas, si l'une des permissions est révoquée par l'utilisateur, toutes les permissions sont révoquées. Si le statut du groupe est « un», cela signifie que seulement l'une des permissions est nécessaire pour assurer la fonctionnalité définie par le But. Cela peut être lié aux données de granularité différentes comme, par exemple, les données de géolocalisation : position exacte, adresse, ville, pays. La fonctionnalité peut être assurée en utilisant l'une de ces données. Si l'une des permissions est autorisée par l'utilisateur, toutes les autres sont révoquées. Dans l'interface utilisateur, cela peut être représenté en temps que menu déroulant pour choisir la granularité des données.

Le système de permissions proposé peut être implémenté en temps que composante intermédiaire entre les applications et le système d'exploitation sur Android. Pour assurer la gestion des fichiers et des dossiers ainsi que le niveau de granularité des différentes données, le nouveau système peut se baser sur un système de gestion de fichiers sémantiques qui permettent de taguer les fichiers. Les appels aux interfaces systèmes issus des applications mobiles doivent passer au travers du système de permissions. Un modèle supplémentaire d'implémentation peut être nécessaire pour assurer la gestion des permissions

sur les données stockées dans les bases des données et sécuriser l'accès aux tables, lignes, colonnes ou cellules.

Un autre axe d'investigation est le travail sur l'affichage des permissions et sur l'interface graphique. L'interface joue un rôle important dans l'adoption d'un tel système par les utilisateurs et aussi dans la compréhension des permissions. Le futur modèle d'interface peut se baser sur un mélange de stratégies existantes comme les pop-up et les tutoriels. L'impact d'intégration d'un tel système sur le processus de développement des applications mobiles doit être évalué. On suppose que l'intégration du système plus respectueux de la vie privée des utilisateurs doit aussi avoir un impact sur le comportement des utilisateurs, le choix des applications, le nombre de téléchargements, etc.

11.6 Conclusion

Cette thèse dresse la problématique du respect et de la sécurité des données privées des utilisateurs mobiles. Dans le cadre de la thèse, nous avons proposé plusieurs modèles pour rendre les applications et les systèmes mobiles plus « Privacy by Design » et pour apporter de la sécurité, de la transparence et du contrôle sur les données. Tout d'abord, on a proposé un patron de conception architecturale pour les développeurs Android. Le patron de conception définit une architecture de l'application et permet aux développeurs de se concentrer sur les fonctionnalités et sur la sécurité en ayant un code structuré, maintenable et extensible. Cette contribution prend en compte le concept « [du] respect de la vie privée inclus de la conception » (en proposant une base claire permettant l'intégration plus facile des autres patrons de conception) et « Sécurité sur le cycle de vie complète» du « Privacy by Design » (en réduisant le nombre de bugs et de failles de sécurité potentielles en clarifiant et structurant le code). Ce travail a été présenté à la conférence internationale PATTERNS 2013 (The Fifth International Conferences on Pervasive Patterns and Applications) dans laquelle nous avons obtenu le prix de « meilleur article » . La version étendue du travail a été publiée dans une revue internationale (International Journal On Advances in Software, volume 7 in 2014).

La deuxième contribution de cette thèse propose une mesure de risque d'une application Android vis-à-vis du respect de données privées ainsi que le seuil du risqué permettant de détecter les applications malveillants et anormales par rapport aux applications dans la catégorie donnée. Comme les utilisateurs ne sont pas capables de juges la sécurité des applications en se basant sur les permissions demandé, Nous avons fait une analyse automatique des demandes des permissions par catégorie pour proposer aux utilisateur un indicateur plus simple. Par rapport aux principes « Privacy by design », cette proposition rend le kiosque de distribution des applications mobiles plus proactifs, sécurisés, transparents et respectueux de la vie privée en indiquant le risque aux utilisateurs avant l'installation des applications. Le motif comportemental propose a été présenté à la conférence internationale PATTERNS 2015 (The Seventh International Conferences on Pervasive Patterns and Applications). La version étendu de se travail est soumis dans le journal Decision Support System Journal, Elsevier.

La dernière proposition de cette thèse est un nouveau système de permission qui prend en compte toutes les notions du « Privacy by Design ». Avec un tel système de permissions, l'utilisateur pourra clairement comprendre laquelle de ses données est utilisée, comment et pour quel but. L'utilisateur aura aussi le contrôle sur ces données. Cette étude a été

validée par la communauté scientifique. La version courte de l'article a été présentée à la conférence internationale PATTERNS 2014 où le prix du meilleur article à été obtenu. La version étendue a été publiée dans la revue internationale (International Journal On Advances in Security). Ce travail a été aussi présenté au 5ème Atelier sur la Protection de la Vie Privée en 2014.

Les travaux de recherche de cette thèse ont été présentés comme poster lors du forum des doctorants organisé par l'Université de Technologie de Troyes en 2015.

Bibliography

[1] TRUSTe : Consumer Mobile Privacy Insights Report, avril 2011.

[2] A Cavoukian : Privacy by design: The 7 foundational principles, 2009.

[3] 15 U.S.C. 6501-6505 : Children's online privacy protection act of 1998, 1988.

[4] Cal. Bus. et Prof. Code §§ 22575-22579 : The online privacy protection act of 2003, 2004.

[5] c. 5) (S.C. 2000 : Personal information protection and electronic documents act, 2015.

[6] Privacy and data protection by design – from policy to engineering. Rapport technique, European Union Agency for Network and Information Security (ENISA), December 2014.

[7] Kamala D Harris : Privacy on the go. *California Department of Justice*, pages 1–27, janvier 2013.

[8] European data protection regulators : Opinion 02/2013 on apps on smart devices. Rapport technique, EU, février 2013.

[9] *Directive 95/46/EC of the European Parliament and of the Council on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data.* European Union, 24 October 1995.

[10] *Proposal for a Regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation).* Council of the European Union, 2015.

[11] *COM(2012) 11 final proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation).* EUROPEAN COMMISSION, 2012.

[12] FRA : Handbook on european data protection law send with email share to google share to del.icio.us share to stumbleupon share to facebook share to twitter handbook on european data protection law, June 2014.

[13] Jaap-Henk Hoepman : Privacy Design Strategies. *CoRR*, octobre 2012.

[14] M Hafiz : A collection of privacy design patterns. *Proceedings of the 13th Conference on Patterns ...*, 2006.

[15] Siani Pearson et Yun Shen : Context-Aware Privacy Design Pattern Selection. *In Trust*, pages 69–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[16] George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Métayer, Rodica Tirtea et Stefan Schiffner : Privacy and Data Protection by Design - from policy to engineering. *CoRR abs/1501.03726*, cs.CR, 2015.

[17] Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides : *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 édition, novembre 1994.

[18] Tuomas Ihme et Pekka Abrahamsson : The Use of Architectural Patterns in the Agile Software Development of Mobile Applications. *In ICAM 2005 Internation Conference on Agility*, pages 155–162, août 2005.

[19] Glenn Krasner et Stephen Pope : A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1:26–49, 1988.

[20] Patrick Sauter, Gabriel Vögler, Günther Specht et Thomas Flor : A Model–View–Controller extension for pervasive multi-client user interfaces. *Personal and Ubiquitous Computing*, 9(2):100–107, mars 2005.

[21] Matthias Veit et Stephan Herrmann : Model-view-controller and object teams: a perfect match of paradigms. *In AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 140–149. ACM Request Permissions, mars 2003.

[22] Steve Burbeck : Applications Programming in Smalltalk-80TM: How to use Model-View-Controller MVC, 1997.

[23] Joëlle Coutaz : PAC. *ACM SIGCHI Bulletin*, 19(2):37–41, octobre 1987.

[24] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad et Michael Stal : *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK, 1996.

[25] M Potel : MVP: Model-View-Presenter the taligent programming model for C++ and Java. *Taligent Inc*, 1996.

[26] J Cai, R Kapila et G Pal : HMVC: The layered pattern for developing strong client tiers, 2000.

[27] Reto Meier : *Professional Android 4 Application Development (Wrox Professional Guides)*. Wrox Press Ltd., Birmingham, 3 édition, mai 2012.

[28] Ivo Salmre : *Writing Mobile Code: Essential Software Engineering for Building Mobile Applications*. Addison-Wesley Professional, février 2005.

[29] S Brahler : Analysis of the android architecture. Rapport technique, Karlsruher Institute of Technology, 2010.

[30] V Rahimian et R Ramsin : Designing an agile methodology for mobile software development: A hybrid method engineering approach. *In Research Challenges in Information Science, 2008. RCIS 2008.*, pages 337–342, 2008.

[31] Pekka Abrahamsson *et al.* : Mobile-D: an agile approach for mobile application development. *In OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 174–175, octobre 2004.

[32] Hyun Jung La et Soo Dong Kim : Balanced MVC Architecture for Developing Service-Based Mobile Applications. *In e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on*, pages 292–299, 2010.

[33] Apostolos Papageorgiou, Bastian Leferink, Julian Eckert, Nicolas Repp et Ralf Steinmetz : Bridging the gaps towards structured mobile SOA. *In MoMM '09: Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, pages 288–294, décembre 2009.

[34] D Plakalovic et D Simic : Applying MVC and PAC patterns in mobile applications. *Journal of Computing*, 2(1):65–72, janvier 2010.

[35] Dimitrios Zissis, Dimitrios Lekkas et Panayiotis Koutsabasis : Design and Development Guidelines for Real-Time, Geospatial Mobile Applications: Lessons from 'MarineTraffic'. *In Mobile Web and Information Systems*, pages 107–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[36] Woon-Yong Kim et Seok-Gyu Park : The 4-tier design pattern for the development of an android application. *Lecture Notes in Computer Science*, 7105:196–203, décembre 2011.

[37] Michael J Jipping : *Smartphone Operating System Concepts with Symbian OS: A Tutorial Guide*. Wiley, Chichester, 2007.

[38] Michael Lane : Does the android permission system provide adequate information privacy protection for end-users of mobile apps? . décembre 2012.

[39] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh et David Wetherall : A Conundrum of Permissions: Installing Applications on an Android Smartphone. *In ... Cryptography and Data ...*, pages 68–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[40] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin et David Wagner : Android permissions: User attention, comprehension, and behavior. *In Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[41] Jagdish Prasad Achara, Mathieu Cunche, Vincent Roca et Aurélien Francillon : WifiLeaks: underestimated privacy implications of the access_wifi_state android permission. *WISEC*, pages 231–236, 2014.

[42] Adrienne Porter Felt, Kate Greenwood et David Wagner : The effectiveness of application permissions. *In WebApps'11: Proceedings of the 2nd USENIX conference on Web application development*, pages 7–7. USENIX Association, juin 2011.

[43] Sunny Consolvo, Ian E Smith, Tara Matthews, Anthony LaMarca, Jason Tabert et Pauline Powledge : Location disclosure to social relations: why, when, & what people want to share. *In CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, avril 2005.

[44] Alina Hang, Emanuel von Zezschwitz, Alexander De Luca et Heinrich Hussmann : Too much information!: user attitudes towards smartphone sharing. *In NordiCHI '12: Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*. ACM, octobre 2012.

[45] Serge Egelman, AdriennePorter Felt et David Wagner : Choice architecture and smartphone privacy: There's a price for that. *In The Economics of Information Security and Privacy*, pages 211–236. Springer Berlin Heidelberg, 2013.

[46] Rainer Böhme et Stefan Köpsell : Trained to accept?: a field experiment on consent dialogs. *In CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, avril 2010.

[47] E Smith : iPhone applications and privacy issues: An analysis of application transmission of iPhone unique device identifiers UDIDs, October 2010.

[48] F Roesner, T Kohno, A Moshchuk, B Parno, H J Wang et C Cowan : User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems. *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 224–238, 2012.

[49] David Barrera, H Güneş Kayacik, Paul C van Oorschot et Anil Somayaji : A methodology for empirical analysis of permission-based security models and its application to android. *In Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 73–84. ACM, octobre 2010.

[50] Pern Hui Chia, Yusuke Yamamoto et N Asokan : Is this app safe?: a large scale study on application permissions and risk signals. *In Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 311–320. ACM, avril 2012.

[51] M Frank, Ben Dong, A P Felt et D Song : Mining Permission Request Patterns from Android and Facebook Applications. *In IEEE International Conference on Data Mining (ICDM)*, pages 870–875, 2012.

[52] I Rassameeroj et Y Tanahashi : Various approaches in analyzing Android applications with its permission-based security models. *In Electro/Information Technology (EIT), 2011 IEEE International Conference on*, pages 1–6, 2011.

[53] William Enck, Machigar Ongtang et Patrick McDaniel : On lightweight mobile phone application certification. *In CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*. ACM, novembre 2009.

[54] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru et Ian Molloy : Using probabilistic generative models for ranking risks of Android apps. *In Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 241–252. ACM, octobre 2012.

[55] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru et Ian Molloy : Android permissions: a perspective combining risks and benefits. *In SACMAT '12: Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. ACM, juin 2012.

[56] N Peiravian et Xingquan Zhu : Machine Learning for Android Malware Detection Using Permission and API Calls. *In Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 300–305, 2013.

[57] Veelasha Moonsamy, Jia Rong, Shaowu Liu, Gang Li et Lynn Batten : Contrasting Permission Patterns between Clean and Malicious Android Applications. *In Future Generation Computer Systems*, pages 69–85. Springer International Publishing, Cham, 2013.

[58] International Secure Systems Lab : Andrubis.

[59] W Wang, X Wang, D Feng, J Liu, Z Han et X Zhang : Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection. *Information Forensics and Security, IEEE Transactions on*, 9(11):1869–1882, 2014.

[60] Wei Xu, Fangfang Zhang et Sencun Zhu : Permlyzer: Analyzing permission usage in Android applications. *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 400–410, 2013.

[61] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song et David Wagner : Android permissions demystified. *In CCS '11: Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, octobre 2011.

[62] Ryan Stevens, Jonathan Ganz, Vladimir Filkov, Premkumar T Devanbu et Hao Chen : Asking for (and about) permissions used by Android apps. *MSR*, pages 31–40, 2013.

[63] Yuvraj Agarwal et Malcolm Hall : ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. *In Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, MobiSys '13, pages 97–110. ACM, juin 2013.

[64] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck et Tao Xie : Whyper: Towards automating risk assessment of mobile applications. *In Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 527–542, Berkeley, CA, USA, 2013. USENIX Association.

[65] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu et Zhong Chen : Autocog: Measuring the description-to-permission fidelity in android applications. *In Proceedings of the 2014 ACM SIGSAC Conference on Computer*

and Communications Security, CCS '14, pages 1354–1365, New York, NY, USA, 2014. ACM.

[66] Hengshu Zhu, Hui Xiong, Yong Ge et Enhong Chen : Mobile app recommendations with security and privacy awareness. *In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14, pages 951–960. ACM, août 2014.

[67] Alastair R Beresford, Andrew Rice, Nicholas Skehin et Ripduman Sohan : MockDroid: trading privacy for application functionality on smartphones. *In HotMobile '11: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54. ACM, mars 2011.

[68] Kurt Mueller et Kevin Butler : Poster: Flex-p: Flexible android permissions. *In IEEE Symposium on Security and Privacy*, 2011.

[69] Yajin Zhou, Xinwen Zhang, Xuxian Jiang et Vincent W Freeh : Taming Information-Stealing Smartphone Applications (on Android). *In Trust and Trustworthy Computing*, pages 93–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[70] I Ion, B Dragovic et B Crispo : Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices. *In Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 233–242, 2007.

[71] Mohammad Nauman, Sohail Khan et Xinwen Zhang : Apex: extending Android permission model and enforcement with user-defined runtime constraints. *In ASIACCS '10: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 328–332. ACM, avril 2010.

[72] M Ongtang, S McLaughlin, W Enck et P McDaniel : Semantically Rich Application-Centric Security in Android. *In Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 340–349, 2009.

[73] Johann Vincent, Christine Porquet, Maroua Borsali et Harold Leboulanger : Privacy Protection for Smartphones: An Ontology-Based Firewall. *In Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, pages 371–380. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[74] Mauro Conti, Vu Thien Nga Nguyen et Bruno Crispo : CRePE: Context-Related Policy Enforcement for Android. *In Information Security*, pages 331–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[75] Jaehong Park et Ravi Sandhu : The UCONABC usage control model. *Transactions on Information and System Security (TISSEC*, 7(1), février 2004.

[76] Srijith K Nair, Andrew S Tanenbaum, Gabriela Gheorghe et Bruno Crispo : Enforcing DRM policies across applications. *In DRM '08: Proceedings of the 8th ACM workshop on Digital rights management*. ACM, octobre 2008.

[77] J Jeon, K K Micinski, J A Vaughan, N Reddy et Y Zhu : Dr. Android and Mr. Hide: Fine-grained security policies on unmodified Android. *Digital Repository at the University of Maryland (DRUM)*, 2011.

[78] Shashank Holavanalli, Don Manuel, Vishwas Nanjundaswamy, Brian Rosenberg, Feng Shen, Steven Y Ko et Lukasz Ziarek : Flow Permissions for Android. *In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 652–657, 2013.

[79] Sven Bugiel, Stephan Heuser et Ahmad-Reza Sadeghi : mytunes: Semantically linked and user-centric fine-grained privacy control on android. Rapport technique TUD-CS-2012-0226, Center for Advanced Security Research Darmstadt, novembre 2012.

[80] Tim Werthmann, Ralf Hund, Lucas Davi, Ahmad-Reza Sadeghi et Thorsten Holz : PSiOS: bring your own privacy & security to iOS devices. *ASIACCS*, pages 13–24, 2013.

[81] T Vidas, N Christin et L Cranor : Curbing android permission creep. *In W2SP*, 2011.

[82] Adam P. Fuchs, Avik Chaudhuri et Jeffrey S. Foster : Scandroid: Automated security certification of android applications, 2009.

[83] M Egele, C Kruegel, E Kirda et G Vigna : PiOS: Detecting Privacy Leaks in iOS Applications. *NDSS*, 2011.

[84] Clint Gibler, Jonathan Crussell, Jeremy Erickson et Hao Chen : AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. *In Trust and Trustworthy Computing*, pages 291–307. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[85] Jagdish Prasad Achara, Franck Baudot, Claude Castelluccia, Geoffrey Delcroix et Vincent Roca : Mobilitics: Analyzing Privacy Leaks in Smartphones. *ERCIM News 2013(93)*, 2013.

[86] J Kim, Y Yoon, K Yi, J Shin et S Center : ScanDal: Static analyzer for detecting privacy leaks in android applications. *MoST*, 2012.

[87] E Chin, A P Felt, K Greenwood et D Wagner : Analyzing inter-application communication in Android. *In Proceedings of the 9th ...*, 2011.

[88] Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein et Yves Le Traon : Effective inter-component communication mapping in Android with Epicc: an essential step towards holistic security analysis. *In SEC'13: Proceedings of the 22nd USENIX conference on Security*. USENIX Association, août 2013.

[89] Patrick P F Chan, Lucas C K Hui et S M Yiu : DroidChecker: analyzing android applications for capability leak. *In WISEC '12: Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, avril 2012.

[90] Christopher Mann et Artem Starostin : A framework for static detection of privacy leaks in android applications. *In Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1457–1462, New York, NY, USA, 2012. ACM.

[91] Feng Shen, Namita Vishnubhotla, Chirag Todarka, Mohit Arora, Babu Dhandapani, Eric John Lehner, Steven Y Ko et Lukasz Ziarek : Information flows as a permission mechanism. *In ASE '14: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, septembre 2014.

[92] Fengguo Wei, Sankardas Roy, Xinming Ou et Robby : Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. *In CCS '14: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, novembre 2014.

[93] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau et Patrick McDaniel : FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. *In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 259–269. ACM, juin 2014.

[94] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel et Anmol N Sheth : TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6. USENIX Association, octobre 2010.

[95] Jagdish Prasad Achara, James-Douglass Lefruit, Vincent Roca et Claude Castelluccia : Detecting privacy leaks in the RATP App: how we proceeded and what we found. *J. Computer Virology and Hacking Techniques ()*, 10(4):229–238, 2014.

[96] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter et David Wetherall : These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. *In Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 639–652. ACM, octobre 2011.

[97] Pascal Berthomé et Jean-François Lalande : Comment ajouter de la privacy after design pour les applications Android? (How to add privacy after design to Android applications?), juin 2012.

[98] Wook Shin, S Kiyomoto, K Fukushima et T Tanaka : Towards Formal Analysis of the Permission-Based Security Model for Android. *In Wireless and Mobile Communications, 2009. ICWMC '09. Fifth International Conference on*, pages 87–92. IEEE Computer Society, 2009.

[99] J Tam, R W Reeder et S Schechter : Disclosing the authority applications demand of users as a condition of installation. Microsoft Research, 2010.

[100] Adrienne Porter Felt, Helen J Wang, Alexander Moshchuk, Steven Hanna et Erika Chin : Permission re-delegation: attacks and defenses. *In SEC'11: Proceedings of the 20th USENIX conference on Security*. USENIX Association, août 2011.

[101] Wook Shin, Sanghoon Kwak, S Kiyomoto, K Fukushima et T Tanaka : A Small But Non-negligible Flaw in the Android Permission Scheme. *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, pages 107–110, 2010.

[102] Sarah Allen, Vidal Graupera et Lee Lundrigan : *Pro Smartphone Cross-Platform Development: IPhone, Blackberry, Windows Mobile and Android Development and Distribution*. Apress, Berkely, 1st édition, septembre 2010.

[103] Kerri Shotts : *PhoneGap for Enterprise*. PACKT Publishing, PhoneGap for Enterprise Mastering Kerri Shotts December 2014 2014.

[104] Abhishek Nalwaya : *Rhomobile Beginner's Guide*. PACKT Publishing, 2011.

[105] Michael Facemire, Jeffrey S. Hammond, Christopher Mines, Dominique Whittaker et Eric Wheeler : The engagement platform's aggregation tier a closer look at the heart of modern enterprise architecture. Rapport technique, Forrester, 2014.

[106] Dave Mark et Jeff LaMarche : *More IPhone 3 Development*. Tackling Iphone Sdk 3. Apress, Berkely, janvier 2010.

[107] James Steele, Nelson To, Shane Conder et Lauren Darcey : *The Android Developer's Collection*. Addison-Wesley Professional, décembre 2011.

[108] Brian Foote et Joseph Yoder : Big Ball of Mud. *In Pattern Languages of Program Design*, pages 653–692. Addison-Wesley, 1997.

[109] Yonghee Shin et Laurie Williams : Is complexity really the enemy of software security? *In Proceedings of the 4th ACM Workshop on Quality of Protection*, QoP '08, pages 47–50, New York, NY, USA, 2008. ACM.

[110] Florent Garin : *Android - Concevoir et développer des applications mobiles et tactiles (Android - Comprehend and develop mobile and tactile applications)*. Dunod, 2nd édition, mars 2011.

[111] Paul D Sheriff : *Fundamentals of N-Tier Architecture*. PDSA Inc., mai 2006.

[112] Steve McConnell : *Tout sur le code : Pour concevoir du logiciel de qualité (Everything about code: make software of quality)*. Dunod, 2nd édition, février 2005.

[113] M T Ionita, D K Hammer et H Obbink : Scenario-based software architecture evaluation methods: An overview. *In Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering*, 2002.

[114] Pkofler : A source measurement suite for java.

[115] Thomas J. McCabe : A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2(2):308–320, 1976.

[116] Karina Sokolova, Marc Lemercier et Ludovic Garcia : Android Passive MVC: a Novel Architecture Model for the Android Application Development. *In* IARIA, éditeur : *PATTERNS 2013, The Fifth International Conferences on Pervasive Patterns and Applications*, pages 7–12, 2013.

[117] Karina Sokolova, Marc Lemercier et Ludovic Garcia : Towards high quality mobile applications: Android passive mvc architecture. *International Journal On Advances in Software*, 7(12):123–138, June 2014.

[118] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna et David Wagner : A survey of mobile malware in the wild. *In the 1st ACM workshop*, pages 3–14, New York, New York, USA, 2011. ACM Press.

[119] Onder Vincent Koc : android-market-api-php.

[120] Linton C. Freeman : Centrality in social networks conceptual clarification. *Social Networks*, page 215, 1978.

[121] Lawrence Page, Sergey Brin, Rajeev Motwani et Terry Winograd : The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[122] Jon M. Kleinberg : Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, septembre 1999.

[123] Andrew P. Bradley : The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):1145–1159, juillet 1997.

[124] Pablo Bermejo, José A. Gámez et Jose Miguel Puerta : Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Syst. Appl.*, 38(3):2072–2080, 2011.

[125] Mathieu Bastian, Sebastien Heymann et Mathieu Jacomy : Gephi: An open source software for exploring and manipulating networks, 2009.

[126] Thomas M. J. Fruchterman et Edward M. Reingold : Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, novembre 1991.

[127] Mathieu Jacomy, Sebastien Heymann, Tommaso Venturini et Mathieu Bastian : Forceatlas2, a continuous graph layout algorithm for handy network visualization. *Medialab center of research*, 560, 2011.

[128] Interactive permissions networks for each category.

[129] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann et Ian H. Witten : The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, novembre 2009.

[130] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero et Pablo Garcia Bringas : On the automatic categorisation of android applications. *In CCNC'12*, pages 149–153, 2012.

[131] Asaf Shabtai, Yuval Fledel et Yuval Elovici : Automated static code analysis for classifying android applications using machine learning. *In Proceedings of the 2010 International Conference on Computational Intelligence and Security*, CIS '10, pages 329–333, Washington, DC, USA, 2010. IEEE Computer Society.

[132] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Javier Nieves, Pablo G. Bringas et Gonzalo Álvarez : Mama: Manifest analysis for malware detection in android. *Cybern. Syst.*, 44(6-7):469–488, octobre 2013.

[133] Marc Lemercier Karina Sokolova, Charles Perez : Android permission usage: a first step towards detecting abusive applications. *In PATTERNS 2015, The Seventh International Conferences on Pervasive Patterns and Applications*, pages 1–7, 2015.

[134] Charles Perez, Babiga Birregah et Marc Lemercier : A smartphone-based online social network trust evaluation system. *Social Network Analysis and Mining*, 3(4):1293–1310, 2013.

[135] Daniel Le Métayer : A Formal Privacy Management Framework. *Formal Aspects in Security and Trust*, pages 162–176, 2008.

[136] Adam Barth, Anupam Datta, John C Mitchell et Helen Nissenbaum : Privacy and Contextual Integrity: Framework and Applications. *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

[137] K Sokolova, M Lemercier et J B Boisseau : Privacy by Design Permission System for Mobile Applications. *In PATTERNS 2014, The Sixth International Conferences on Pervasive Patterns and Applications*, pages 89–95, 2014.

[138] K Sokolova, M Lemercier et J B Boisseau : Respecting user privacy in mobiles: privacy by design permission system for mobile applications. *International Journal On Advances in Security*, 7(34):110–120, December 2014.

Karina SOKOLOVA PEREZ

Doctorat : Ingénierie Sociotechnique des Connaissances, des Réseaux et du Développement Durable

Année 2016

Modèles pour les environnements de terminaux nomades « Privacy by Design »

De nos jours, les smartphones et les tablettes génèrent, reçoivent, mémorisent et transfèrent vers des serveurs une grande quantité de données en proposant des services aux utilisateurs via des applications mobiles facilement téléchargeables et installables. Le grand nombre de capteurs intégrés dans un smartphone lui permet de collecter de façon continue des informations très précise sur l'utilisateur et son environnement. Cette importante quantité de données privées et professionnelles devient difficile à superviser.

L'approche «Privacy by Design», qui inclut sept principes, propose d'intégrer la notion du respect des données privées dès la phase de la conception d'un traitement informatique. En Europe, la directive européenne sur la protection des données privées (Directive 95/46/EC) intègre des notions du «Privacy by Design». La nouvelle loi européenne unifiée (General Data Protection Régulation) renforce la protection et le respect des données privées en prenant en compte les nouvelles technologies et confère au concept de «Privacy by Design» le rang d'une obligation légale dans le monde des services et des applications mobiles.

L'objectif de cette thèse est de proposer des solutions pour améliorer la transparence des utilisations des données personnelles mobiles, la visibilité sur les systèmes informatiques, le consentement et la sécurité pour finalement rendre les applications et les systèmes mobiles plus conforme au «Privacy by (re)Design».

Mots clés : design patterns - systèmes informatiques, mesures de sûreté - Google Android (système d'exploitation des ordinateurs) - droit à la vie privée - smartphones, accès, contrôle.

Bridging the Gap between Privacy by Design and Mobile Systems by Patterns

Nowadays, smartphones and smart tablets generate, receive, store and transfer substantial quantities of data, providing services for all possible user needs with easily installable programs, also known as mobile applications. A number of sensors integrated into smartphones allow the devices to collect very precise information about the owner and his environment at any time. The important flow of personal and business data becomes hard to manage.

The “Privacy by Design” approach with 7 privacy principles states privacy can be integrated into any system from the software design stage. In Europe, the Data Protection Directive (Directive 95/46/EC) includes “Privacy by Design” principles. The new General Data Protection Regulation enforces privacy protection in the European Union, taking into account modern technologies such as mobile systems and making “Privacy by Design” not only a benefit for users, but also a legal obligation for system designers and developers.

The goal of this thesis is to propose pattern-oriented solutions to cope with mobile privacy problems, such as lack of transparency, lack of consent, poor security and disregard for purpose limitation, thus giving mobile systems more Privacy by (re) Design.

Keywords: software patterns - electronic data processing departments, security measures – smartphones, access control - privacy, right of - Google Android (operating system).

Thèse réalisée en partenariat entre :

anrt association nationale recherche technologie

EUTECH SSII the software factory

utt université de technologie Troyes

Ecole Doctorale "Sciences et Technologies"