Université de Montréal

Caractérisation et étude de l'impact des permissions dans les applications mobiles

par
Selsabil Dbouba

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures
en vue de l'obtention du grade de Maitrise sciences (M.Sc.)
en informatique

Décembre, 2018

Résumé

Android fournit un modèle de sécurité basé sur les permissions aux développeurs d'applications tiers, qui contrôlent l'accès aux ressources système, incluant le matériel, les paramètres et les données utilisateur. Cette recherche examine un ensemble de données de permissions pour plus de 130 000 applications existantes sur le *Google Play Store* sur la manière dont ils peuvent affecter les utilisateurs de *smartphones*. L'objectif est d'identifier des patrons d'utilisation de permissions et de mieux comprendre les modèles dans les permissions du point de vue des utilisateurs. Nous rapportons des résultats sur des milliers d'applications et nous observons une corrélation négative entre le score de conformité aux patrons que nous avons calculé pour chaque application, la note moyenne et le prix.

Ce projet présente également une étude empirique du modèle de permission d'***Android***, permettant d'observer l'utilisation de jeux de permissions par des développeurs tiers dans des applications similaires et d'identifier les niveaux de risque associés aux différentes combinaisons de permissions dans les applications ***Android***. Nous avons trouvé que la permission d'accès à Internet a été utilisée par plus de 70% d'applications et que les deux permissions les plus communes qui accèdent aux informations de l'utilisateur sont ACCESS_*FINE_LOCATION* et *ACCESS_COARSE_LOCATION*.

L'étude a démontré que le score de conformité n'est pas un indicateur de risque et il n'est pas non plus un indicateur de pérennité. Par contre le degré d'utilisation de permissions dangereuses est à la fois un indicateur de risque et un indicateur de pérennité, en effet, les applications qui demandent plus de permissions dangereuses ont plus de chance d'être des applications malicieuses et d'être supprimées du marché *Google Play*. De même, nous avons trouvé que la satisfaction des utilisateurs reflète à la fois le risque et la pérennité. En effet, les applications moins notées ont plus de chance d'être des applications malicieuses et d'être supprimées du marché *Google Play*.

Mots-clés : permissions Android, Risque, données personnelles, applications Android, sécurité basée sur les permissions, systèmes d'exploitation de smartphone, Liste de contrôle d'accès.

Abstract

Android provides a security model based on permissions to third-party application developers, who control access to system resources, including hardware, settings, and user data. This search examines a set of permission data for more than 130,000 existing apps on the Google Play Store on how they can affect smartphone users. The goal is to identify patterns of permissions usage and to better understand the patterns in permissions from the users' point of view. We report results on thousands of applications and we observe a negative correlation between the score of conformance to the patterns that we calculated for each application, the average rating and the price.

This project also presents an empirical study of the ***Android*** permission model, allowing to observe the use of permission sets by third-party developers in similar applications and to identify the risk levels associated with different combinations of permissions in the Android applications. We found that Internet access permission was used by over 70% of applications and that the two most common permissions that access user information are ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION.

The study showed that the conformance score is not a risk indicator and is not an indicator of sustainability. On the other hand, the degree of use of dangerous permissions is at the same time a risk indicator and an indicator of durability, in fact, the applications which request more dangerous permissions are more likely to be malicious applications and to be deleted from Google Play store. Likewise, we've found that user satisfaction reflects both risk and sustainability, as less-rated apps are more likely to be malicious apps and removed from the Google Play store.

Keywords: **Android permissions, Risk, Personal data, Android applications, permission-based security, smartphone operating systems, Access control List.**

Table des matières

Liste des tableaux

Liste des figures

Liste des abréviations

API : Application Programming Interface

JNI : Java Native Interface

LDA : Latent Dirichlet Allocation

NDK: Native Development Kit Android

SOM: Self-Organizing Map

SDK: Software Development Kit

SSL: Secure Sockets Layer

URL: Uniform Resource Locator

XML: Extensible Markup Language

À mon cher conjoint Rafik

Remerciements

Je tiens en premier lieu à remercier Professeur Houari Sahraoui et Professeure Esma Aïmeur d'avoir eu la gentillesse de m'accepter au sein de leurs équipes et de m'avoir encadré tout au long de ma formation. La réalisation de ce travail n'aurait pas été possible sans leur soutien, leurs précieux conseils ainsi que leur disponibilité au cours de ce projet.

Je tiens également à adresser mes remerciements à tous les membres du laboratoire de génie logiciel GEODES qui ont permis de créer un environnement de travail amical, convivial et très favorable à l'aboutissement de ce projet de mémoire et spécialement mon cher collègue Mohamed Aymen Saied pour ses contributions et ses outils qui ont permis de réaliser mon projet de mémoire.

Enfin, je remercie ma mère et ma sœur pour leur soutien moral et affectif et mon conjoint Rafik pour son amour et sa grande patience envers mes angoisses de fin de maitrise. Merci d'être toujours là, à côté de moi.

Et pour finir, je tiens à dédier ce travail à la mémoire de mon père, décédé trop tôt, récemment, au cours de mes études, qui m'a toujours encouragée et motivée.

J'espère que, du monde qui est le sien maintenant, il apprécie cet humble geste comme preuve de reconnaissance de la part de sa fille qui a toujours prié pour le salut de son âme, Puisse Dieu, le tout puissant, l'avoir en sa sainte miséricorde !

CHAPITRE 1

INTRODUCTION

1.1 Contexte

De nos jours, les téléphones intelligents sont omniprésents et intégrés dans notre vie quotidienne. Ces dernières décennies, nous assistons à une évolution technologique considérable notamment pour les téléphones mobiles. Autrefois, les téléphones cellulaires servaient principalement à recevoir et à émettre des appels et de courts messages. Au fil des années, les téléphones cellulaires ont rapidement évolué pour nous permettre, une utilisation beaucoup plus efficace et globale en matière de communication et de gestion de l'information. Les téléphones présents aujourd'hui sur le marché sont d'un point de vue technologique, davantage comparables à des micro-ordinateurs offrant une vaste gamme de services tels que gestion de courriels, navigation Internet, achat en ligne, accès au compte bancaire, etc.

Ces dernières années, la gamme de services offerts par les téléphones mobiles a considérablement augmenté en raison de la croissance du nombre d'applications innovantes disponibles sur le marché mobile. Selon les prédictions faites par *Gartner*, les applications mobiles généreront un massif de 77 milliards de dollars, les applications populaires étant téléchargées plus de 268 milliards de fois en 2017 [1].

Android est le système d'exploitation mobile le plus utilisé dans le monde avec 82.8% de parts de marché dans les *smartphones*. Son concurrent le plus proche est ***Apple***, avec 13,9% du marché [2]. Le marché des applications sans restriction d'***Android*** et son système *open source* font de lui une plate-forme populaire pour les applications tierces. Depuis 2011, ***Android Market*** comprend plus d'applications qu'***Apple Store*** [3]. Dans la même tendance, le nombre d'attaques ciblant les appareils et les applications ***Android*** a augmenté en exploitant les vulnérabilités de ces derniers.

1.2 Problématique

Les nouvelles versions de systèmes d'exploitation d'***Android*** ont été développées en accordant plus d'attention à la sécurité et à la protection de la vie privée. ***Android*** utilise un système de permission pour contrôler les privilèges d'accès aux informations sensibles des applications. Ce mécanisme exige que les développeurs déclarent quelles ressources sensibles leurs applications utilisent. Lors de l'installation d'une application, l'utilisateur est informé des permissions requises par celle-ci. Il doit fournir son accord à cette liste de permissions et ainsi il limite l'accès de l'application durant l'exécution seulement aux ressources autorisées [4].

Ces systèmes de permissions sont destinés à aider les utilisateurs à éviter les applications invasives. Malheureusement, plusieurs recherches d'utilisateurs ont montré que de nombreux utilisateurs ne prêtaient pas attention ou ne comprenaient pas les avertissements de permission [5]. Porter Felt et al. (2012) ont effectué deux études d'utilisabilité : un sondage sur Internet de 308 utilisateurs d'***Android*** et une étude de laboratoire dans laquelle ils ont interrogé et observé 25 utilisateurs d'***Android*** [5]. Les participants à l'étude ont montré un faible taux d'attention et de compréhension : l'enquête sur Internet et l'étude en laboratoire ont révélé que 17% des participants ont accordé une attention particulière aux permissions pendant l'installation, et seulement 3% des répondants au sondage sur Internet ont répondu correctement à toutes les trois questions de compréhension de la permission. Cela indique que les alertes de permissions actuelles d'***Android*** ne permettent pas à la plupart des utilisateurs de prendre des décisions de sécurité correctes. Plusieurs études ont examiné l'utilisation des permissions par les applications ***Android***. Felt et al. (2011) ont classé manuellement un petit ensemble d'applications ***Android*** comme étant soit privilégiée ou soit normal, une application est considérée privilégiée si elle accède à une quantité inutile du matériel, des paramètres et des données utilisateur du téléphone. Les chercheurs ont constaté que l'une des principales menaces posées par les applications ***Android*** malveillantes est des violations de la vie privée qui transmettent des informations sensibles telles que des informations de localisation, des données de contact, des images, des messages SMS, etc. à l'attaquant [6].

Barrera et al. (2010) ont analysé les permissions demandées par 1, 100 applications ***Android*** gratuites. Ils se sont concentrés principalement sur la structure du système de permission, ils ont regroupé les applications en utilisant un réseau neuronal et ont recherché des patrons dans les demandes de groupes de permissions. Ils ont noté que 62% des demandes collectées en décembre 2009 utilisent la permission de l'INTERNET [7].

La problématique que nous traitons dans ce mémoire est liée d'une part à l'absence d'un système de vérification intégré pour s'assurer que les applications ne demandent pas d'autorisations inutiles, ce qui augmente la surface d'attaque et rend les applications plus sensibles aux problèmes de sécurité. D'autre part, les développeurs ne respectent pas toujours les règles de bonnes pratiques fourni par Android ce qui rendent les applications plus sensibles aux problèmes de sécurité.

Nos objectifs principaux étaient de démystifier l'utilisation du système de permission dans différentes catégories d'applications Android et d'étudier le risque potentiel que ces applications soient nocives. Bien qu'il existe de nombreux systèmes largement déployés qui utilisent des permissions, nous nous concentrons sur l'analyse empirique du modèle d'autorisation inclus dans Android OS. Les principaux objectifs de notre analyse empirique sont les suivants : étudier comment le système de permission sous Android est utilisé dans la pratique et identifier les points forts et les limitations de la mise en œuvre actuelle. Nous pensons qu'une telle analyse peut révéler des modèles d'utilisation intéressants, en particulier lorsque le système basé sur les permissions est utilisé par un large éventail d'utilisateurs ayant des niveaux d'expertise différents. Nous pensons également que la combinaison des permissions peut rendre les applications invasives. Bien que deux permissions puissent être inoffensives lorsqu'elles sont accordées séparément, les risques de confidentialité et de sécurité peuvent augmenter considérablement si elles sont accordées ensemble. À titre d'exemple, nous prenons les permissions INTERNET et READ SMS, séparément, ils sont inoffensifs, mais combinées les unes avec les autres, nous obtenons une application qui peut lire les messages textes et envoyer le contenu à un tiers. Il existe plusieurs combinaisons de permissions qui comportent des risques importants

mais ne sont pas présentés comme tels à l'utilisateur. Le but de notre projet de mémoire est de réaliser une étude en considérant ces risques.

1.3 Contribution

Ce mémoire a pour objectif d'étudier le lien entre les permissions demandées par les applications ***Android*** et le risque associé à l'utilisation de ces applications. Nous allons explorer d'abord l'existence de patrons d'utilisation de permissions dans certaines catégories d'applications. Pour cela nous allons utiliser une technique de *Clustering* pour extraire ces patrons. Par la suite, nous allons mesurer la conformité des applications aux patrons. Cette conformité va nous permettre de vérifier si oui ou non les applications de telle ou telle catégorie suivent un modèle prédéfini de permissions.

La deuxième contribution de ce mémoire porte sur l'étude des liens entre les permissions et le risque. Dans une étape préliminaire, nous allons étudier la relation entre certaines caractéristiques des applications (coût et évaluation des utilisateurs) et le risque. Par la suite, nous allons nous intéresser au lien entre les permissions et le risque d'utilisation des applications. Ceci se fera en deux étapes. Dans une première phase, nous nous intéressons à la quantité ou le nombre de permissions dangereuses. Par la suite, nous étudierons l'impact de déviation des patrons de permissions sur le risque.

Plus de 170 permissions pour plus de 130 000 applications sur ***Android Market*** ont été collectées et réparties selon les catégories. Celles-ci sont étudiées pour déterminer les inférences à partir des données de l'utilisateur. Dans le cadre de cette étude, nous avons retenu quatre catégories : **Météo**, **Outils**, **Communication**, **Finance**. Ces catégories totalisent respectivement un nombre variable de jeux de données soit 805, 1735, 1992 et 2798 applications. Pour déterminer le risque des applications, nous avons utilisé un outil préexistant qui permet de détecter certaines catégories de *malware.* Nous nous sommes aussi intéressés à la pérennité de ces applications dans le *Google Play Store.* En effet les données que nous avons utilisées dans notre étude ont été collecté en 2012 et nous avons effectué une vérification sur l'ensemble des applications en Juin 2017 afin de valider la disponibilité de l'application dans le marché Google Play. Nous avons trouvé que certaines

applications ont été supprimé du Google Play Store, nous considérons ces applications comme des applications non pérennes et nous nous intéressons à expliquer pourquoi ces applications ont été supprimé car nous pensons que la disponibilité ou l'absence d'une application du Google Play Store a un rapport avec le risque associé à cette application.

1.4 Structure du mémoire

La suite de ce mémoire est organisée de la manière suivante : le chapitre 2 présente une vaste revue de la littérature expliquant le contexte d'***Android***, incluant la sécurité et l'architecture du système d'exploitation ***Android***, les conceptions d'applications et le modèle de permission basé sur la sécurité d'***Android***. Le chapitre 3 constitue le cœur de ce mémoire. Après un aperçu général des notions de base, nous y décrivant en détail, l'algorithme de *Clustering* utilisé dans l'approche proposée et nous présentons une étude de cas concernant les patrons d'utilisations de permissions. Dans le chapitre 4, nous présentons les données recueillies au cours du projet. Nous fournissons un aperçu des permissions analysées à partir des applications échantillonnées et nous présentons notre étude empirique réalisée dont le but d'étudier la relation entre les permissions et le risque. Enfin, le 5e et dernier chapitre conclut le présent mémoire en identifiant notamment certains axes d'amélioration et travaux futurs.

CHAPITRE 2

ÉTAT DE l'ART

Le chapitre 2 fournit un aperçu du système d'exploitation ***Android*** et de ses applications, couvrant les aspects historiques et techniques, incluant les architectures individuelles et le contexte technique. Ce chapitre porte spécifiquement sur l'état actuel de la sécurité des applications ***Android***, les menaces pesant sur les appareils mobiles et la recherche afin d'atténuer ces vulnérabilités. Il est organisé en deux parties. Dans la première partie, nous introduisons les généralités du système d'exploitation mobile ***Android*** et nous explorons dans la deuxième partie les travaux reliés aux permissions dans les applications mobiles.

2.1 Généralités

2.1.1 Architecture du système d'exploitation Android

Android est un système d'exploitation mobile basé sur **le noyau *Linux*** et développé actuellement par ***Google*** lancé en juin 2007. En 2015, ***Android*** est le système d'exploitation mobile le plus utilisé dans le monde avec 82.8% de parts de marché. ***Apple*** vient en deuxième position avec une part de marché de 13,9% et ***Microsoft*** en troisième position avec une part de marché de 2,6% [2]. La domination du marché des téléphones intelligents montre qu'***Android*** est extrêmement populaire.

Android est distribué en *Open Source* sous ***licence Apache***. Le système d'exploitation ***Android*** est basé sur ***Linux*** conçu pour les *smartphones* et les tablettes tactiles. Avec le concept central d'une communauté *open source*, ***Android*** offre ses outils, ses suites et son code source gratuitement. Pour les développeurs, cela permet l'innovation dans la construction d'applications utilisant les dernières technologies mobiles, incluant le codage de langages tels que ***Java*** et ***Dalvik byte code*** (.dex), et un format byte code conçu uniquement pour ***Android*** [8]. Cela fournit un produit qui peut être adapté et personnalisé,

tout en recevant des contributions de la communauté, des développeurs et des utilisateurs d'***Android***.

Android est conçu sous la forme d'**une pile de logiciels** comprenant des applications, un système d'exploitation, un environnement d'exécution, un *middleware*, des services et des bibliothèques. Cette architecture peut, peut-être, être mieux représentée visuellement comme indiqué dans la figure 1. Chaque couche de la pile et les éléments correspondants au sein de chaque couche sont étroitement intégrés et soigneusement accordés pour fournir un environnement de développement et d'exécution optimal pour les appareils mobiles. Le sens de lecture se fait de bas en haut.

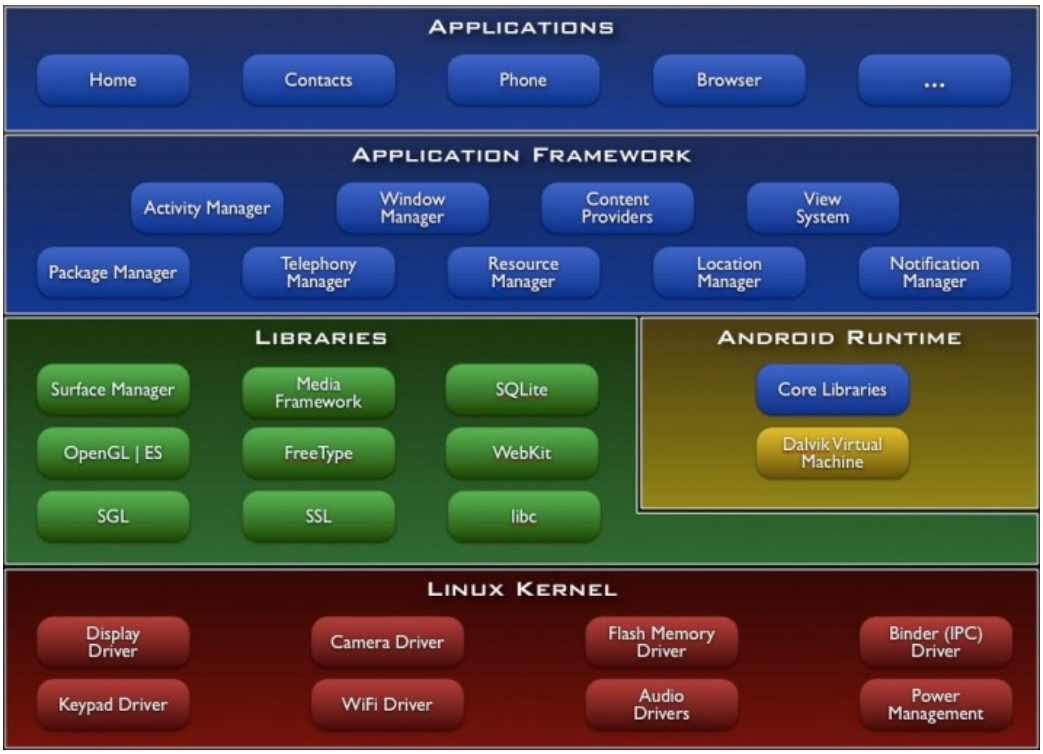

Figure 1 - L'architecture du système Android. Les éléments verts sont écrits en C / C ++, les éléments bleus sont écrits en Java et exécutés dans la VM Dalvik [9].

2.1.2 Android Applications

2.1.2.1 Principe de base et composants d'une application Android

Une application ***Android*** est écrite en ***Java*** et compilée dans un seul fichier d'archive à l'aide de la trousse d'outils **SDK d'*Android***. Cela prend le code *Java* et le compile dans un fichier de package ***Android***, appelé **APK** (*.apk*). Les fichiers *APK* contiennent toutes les ressources et les données pertinentes requises pour installer et exécuter une application sur un appareil ***Android***. Le diagramme illustré dans la figure 2 représente les composants impliqués dans la construction et l'exécution d'une application :

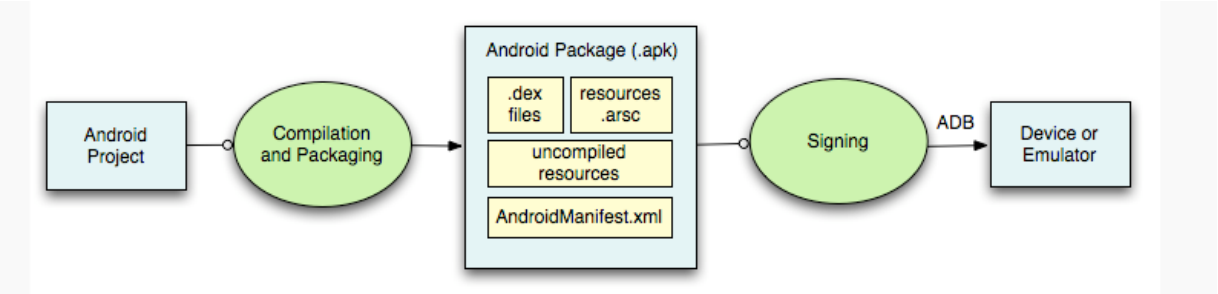

Figure 2 - Construction et exécution d'une application Android [12]

Les composants d'une application sont les blocs de construction initiaux d'une application ***Android***. Chaque composant a un rôle différent associé au système et éventuellement à l'utilisateur. Les quatre composants principaux des applications ***Android*** sont ***Activity***, ***Content Provider***, ***Service*** et ***Broadcast Receiver***.

Activity **:** Toutes les applications ***Android*** sont composées d'activités, qui sont des tâches autonomes conçues pour fournir des informations à l'utilisateur. Les activités représentent habituellement un seul écran d'information, comme la composition d'un message instantané ou d'un courrier électronique. Les activités sont proactives et commencent et s'annulent mutuellement, créant un flux d'informations clair sur l'interface utilisateur, y compris les activités d'appel d'autres applications installées sur l'appareil.

Content Provider **:** Un *Content Provider* est un composant qui sert à gérer les données associées à une application, partagée ou privée. Ces données peuvent être stockées dans un système de fichiers, une base de données *SQLite*, sur le *Web* ou tout autre emplacement de

stockage persistant. Les applications font appel au *content provider* pour consulter ou même modifier les données stockées (uniquement avec l'autorisation du *content provider* des applications). Par exemple, les informations de contact comme les noms et les nombres peuvent être utilisés de manière dynamique sur une grande variété d'applications ***Android***, un *Content Provider* est mis en place pour distribuer et contrôler la modification et le flux de ces données.

Service* :** Le composant Service des applications ***Android n'a pas d'interface utilisateur et est responsable de l'exécution de tous les processus d'arrière-plan des applications, y compris l'extraction de données à distance sur un réseau alors qu'un utilisateur interagit avec d'autres applications sur leur appareil. Les services sont généralement appelés via une activité pour fournir d'autres fonctionnalités à une application.

Broadcast Receiver* :** Un *Broadcast Receiver* est responsable d'annoncer des notifications à l'échelle du système. Ce composant n'est pas associé à une interface graphique utilisateur, cependant, il peut toujours annoncer des notifications telles que la batterie faible et d'autres messages système à l'utilisateur. En ce qui concerne les applications ***Android, *Broadcast Receiver* permettra à une application de diffuser des notifications à d'autres applications, y compris les nouvelles initiations de données, activités ou services. Référé en tant que passerelle à d'autres composants, le *Broadcast Receiver* est destiné à effectuer le moins de travail des quatre composants de l'application.

2.1.2.2 Intent : communication entre composants

Android utilise un objet de messagerie asynchrone appelé *Intent* pour activer ces composants individuels et leur permettre d'envoyer des messages et des commandes les uns aux autres. Nous pouvons considérer *Intent* comme un messager qui demande une action à partir d'un autre composant. Un *Intent* sera écrit pour décrire la tâche demandée (activité, service, fournisseur ou diffusion) et les données nécessaires pour accomplir la tâche [13]. Ces messages seront transmis en arrière et en avant entre les quatre composants à plusieurs reprises, pour construire la plus grande image d'une application fonctionnelle en marche.

Un *Intent* peut avoir plusieurs attributs dont quatre principaux : *component Name*, *action*, *data* et *category*. Ces attributs servent à définir le(s) destinataire(s) de l'*Intent* et les données à transmettre. *Component Name* désigne le nom du composant destiné à recevoir le message. Si une valeur est associée à cet attribut, l'*Intent* est dit explicite. Dans le cas contraire, il est dit implicite et il appartient au système de définir le destinataire du message. S'il existe plusieurs destinataires possibles, le système demandera à l'utilisateur de choisir le destinataire à qui le message sera envoyé. Ce mécanisme est appliqué uniquement lorsque les destinataires possibles sont de type *Activity*. S'il s'agit de *Service*, le système fera lui-même le choix. La figure 3 présente un processus d'*Intent* implicite simple. L'activité A a créé un *Intent* avec une description de l'action et le transmet à *startActivity* (). Le système ***Android*** recherche dans toutes les applications un filtre d'*Intent* qui correspond à la demande d'*Intent* envoyée par A. Lorsqu'une correspondance est trouvée, le système démarre l'activité correspondante (activité B) en invoquant sa méthode *onCreate* () et en lui transmettant l'*Intent*.

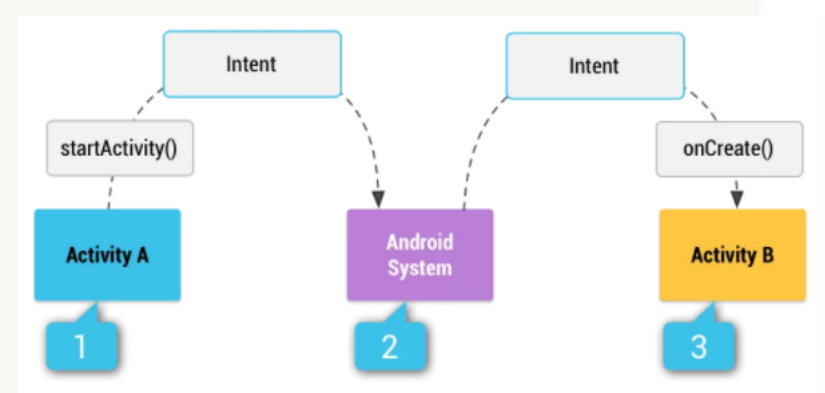

Figure 3 - Illustration de la façon dont un Intent est délivré par le système pour commencer une autre activité [14]

Deux autres attributs peuvent s'ajouter aux quatre précédents : les *flags* et les *extras*. Le système ne les utilise pas pour définir le ou les destinataire(s) de l'*Intent*. Les *flags* sont des attributs destinés plus au système qu'au destinataire du message. Une analyse rapide de la liste des *flags* fournis par ***Android*** montre qu'ils ont un usage assez varié. Ils servent par exemple à déléguer une permission d'accès en lecture ou écriture aux données associées

à l'*Intent* ou à choisir des groupes d'applications pouvant recevoir le message. Les *extras* permettent de transmettre des informations additionnelles au destinataire du message.

2.1.2.3 Android package

Android Package est la forme sous laquelle une application est proposée à l'utilisateur. Il s'agit d'une archive, communément appelée ***apk***, contenant le code de l'application et les ressources qu'elle utilise. Nous présentons dans ce qui suit le contenu principal d'un *apk* :

- **AndroidManifest.xml** : *AndroidManifest.xml* est un fichier *XML* contenant les informations liées à l'application qui sont nécessaires au système. Il est créé par le développeur de l'application. Parmi les informations qu'il contient, nous pouvons citer :
    - Le nom du package de l'application,
    - Le composant *Activity* à lancer au lancement de l'application,
    - La liste des composants de l'application et les informations qui sont liées aux composants (ex : permission pour accéder à un composant *Service* sensible, les *Intents* attendus par les composants de l'application, etc.)
    - Les permissions déclarées par l'application (ex : *write_sms* pour écrire les SMS).
    - Les bibliothèques utilisées par l'application.
    - Le niveau minimal du *SDK* ***Android*** pour que l'application puisse fonctionner.

La figure 4 présente le fichier *AndroidManifest.xml* d'une version d'*Angry Birds Space*. À la deuxième ligne sont définis l'endroit où l'application sera installée (ici laissé au choix du système) et le nom du package de l'application. De la ligne 4 à la ligne 11 sont ensuite listées les permissions demandées par l'application. Ici l'application demande par exemple l'accès à la mémoire externe, au réseau, aux informations liées au téléphone, aux données de géolocalisation et au système de payement proposé par *Google* [15].

La ligne 13 décrit comment l'application sera présentée dans le menu des applications une fois installée : l'icône à utiliser et le nom à afficher. Le reste du fichier liste les composants de l'application et les informations qui y sont liées. L'application a ainsi quatre composants dont deux de type *Activity*, un de type *Service* et un de type *Broadcast Receiver*. Comme expliqué précédemment, filtrer les messages *Intent* à destination des composants de l'application est possible. Dans *AndroidManifest.xml*, cela se fait via les entrées *intent-filter*. Ainsi le premier *Activity* ne recevra que les *Intents* dont l'attribut action a la valeur *android.intent.action.MAIN* et la catégorie est *android.intent.category.LAUNCHER*. Cela signifie que c'est ce composant qui sera lancé lorsque l'utilisateur cliquera sur l'icône de l'application dans le menu de son téléphone. Quant au Broadcast Receiver, il ne réagira qu'aux intents dont l'attribut action vaut *com.android.vending.billing.RESPONSE_CODE*.

- ***Classes.dex*** : Le code de chaque classe d'une application ***Java standard*** est stocké dans des fichiers .class différents. Sous ***Android***, ce n'est pas le cas. Tout est stocké dans un seul et unique fichier qui est *Classes.dex*. De plus, si le code des applications *Java* est lui compilé en *byte code Java*, celui des applications ***Android*** est lui compilé dans un autre format qui est le *byte code Dalvik*. C'est le contenu de ce fichier, ou plus précisément une version optimisée de celui-ci, qui sera interprété par la machine virtuelle Dalvik pour exécuter l'application.
- **Autres** : Un *apk* contient d'autres entrées telles que les répertoires **META-INF**, **res**, **jni** et **lib**. Le répertoire *META-INF* contient ainsi des fichiers liés au contrôle d'intégrité de l'application et à l'identification de son développeur. Le répertoire *res* contient les ressources utilisées par l'application telle que des images, sons, etc. Les répertoires *jni* et *lib* contiennent les bibliothèques utilisées par l'application.

```
<?xml version="1.0" encoding="utf-8"?> <manifest
    android:installLocation="auto"
    package="com.rovio.angrybirdsspace.ads"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE"
        /> <uses-permission
        android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="android.permission.ACCESS_WIFI_STATE"
        /> <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION"
        /> <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE"
        /> <uses-permission
        android:name="android.permission.READ_PHONE_STATE"
        /> <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        /> <uses-permission
        android:name="com.android.vending.BILLING" />

    <application android:label="Angry Birds"
        android:icon="@drawable/icon"> <activity
            android:name="com.rovio.fusion.App"
            android:launchMode="singleTask"
            android:screenOrientation="landscape">
            <intent-filter> <action
                    android:name="android.intent.action.MAIN"
                    /> <category
                    android:name="android.intent.category.LAUNCHER"
                    /> </intent-filter> </activity>

        <service
            android:name="com.rovio.fusion.GooglePlayInAppBilling.
            BillingService"/>

        <receiver
            android:name="com.rovio.fusion.GooglePlayInAppBilling.
            BillingReceiver">
            <intent-filter> <action android:name="com.android.
                    vending.billing.RESPONSE_CODE"/>
            </intent-filter> </receiver>

        <activity
            android:name="com.rovio.fusion.MediaPlayerWrapper"
            android:screenOrientation="behind"
            android:configChanges="keyboardHidden|orientation|screenSize"
            /> </application> <uses-feature
        android:glEsVersion="0x20000"
        android:required="true" /> </manifest>
```

Figure 4 - Fichier AndroidManifest.xml d'Angry Birds Space [15]

2.1.3 Modèle de permission Android

Le système d'exploitation ***Android*** est un système ***Linux*** **multi-utilisateurs** dans lequel chaque application est associée à un compte utilisateur (user) diffèrent, le système attribue par défaut, à chaque application un User-ID (UID) unique et non connu de l'application. En fait, chaque application s'exécute avec une identité de système distincte et est isolée par le noyau Linux dans un *sandbox* virtuel, en les séparant d'autres applications pour protéger l'intégrité du système et la confidentialité de l'utilisateur. Chaque application accède par défaut uniquement aux composants nécessaires à son exécution. En fait, si une application veut utiliser des ressources ou des informations en dehors de son *sandbox*, elle doit demander explicitement l'autorisation. En fonction du type de permission demandée par l'application, le système peut l'accorder automatiquement ou le système peut demander à l'utilisateur d'accorder la permission [16].

Une application ***Android*** de base ne possède aucune permission associée par défaut, les développeurs sont responsables du remplissage d'une application avec des permissions, adaptées à ses fonctionnalités. Précédemment discutées dans la section 2.1.2.2, les permissions d'applications sont stockées dans un fichier appelé « *AndroidManifest.xml* » situé dans le fichier « *Android Package* (*.apk*) » de l'application.

La balise <*uses-permission*> est un élément du fichier *AndroidManifest.xml* qui permet aux développeurs de déclarer quelles permissions doivent être accordées pour que l'application fonctionne correctement. Les permissions sont accordées par l'utilisateur lorsque l'application est installée sur des appareils fonctionnant sous ***Android* 5.1** ou lorsque l'application est exécutée sur des appareils exécutant ***Android* 6.0** ou supérieurs. La figure ci-dessous présente un exemple de la balise <*uses-permission*> demandant la permission d'écrire sur le stockage externe.

```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18" />
```

Figure 5 - Un exemple de balise XML <use-permission> [17]

Les permissions d'***Android*** sont classées en quatre niveaux de protection, à savoir **Normal**, **Dangereuse**, **Signature**, et **SignatureOrSystem**. Le tableau 1 présente les significations définies par ***Android*** derrière chaque niveau de protection de permission [18].

Tableau 1 - Niveau de protection de permission

Niveau de protection	**Description**
Normal	Une permission de **faible risque** qui permet aux applications demandant d'accéder à des fonctionnalités de niveau d'application isolées, avec un risque minimal pour les autres applications, le système ou l'utilisateur. Le système accorde automatiquement ce type de permission à une application requérante lors de l'installation, sans demander l'approbation explicite de l'utilisateur (bien que l'utilisateur ait toujours l'option de passer en revue ces permissions avant l'installation).
Dangereuse	Une permission à **risque supérieur** qui donnerait à une application requérante un accès aux données utilisateurs privés ou un contrôle sur le périphérique qui pourrait avoir un impact négatif sur l'utilisateur. Parce que ce type de permission introduit un risque potentiel, le système ne peut pas

	l'accorder automatiquement à l'application demandeuse. Par exemple, toutes les permissions dangereuses demandées par une application peuvent être affichées à l'utilisateur et nécessitent une confirmation avant de procéder, ou une autre approche peut être prise pour éviter que l'utilisateur autorise automatiquement leurs installations.
Signature	Une permission que le système accorde que si l'application qui la demande est signée avec le même certificat que l'application qui a déclaré la permission.
SignatureOrSystem	Une permission que le système accorde uniquement aux applications qui se trouvent dans l'image système ***Android*** ou qui sont signées avec le même certificat que l'application qui a déclaré la permission. De préférence, il faut éviter d'utiliser cette option, car le niveau de protection de la signature devrait être suffisant pour la plupart des besoins et fonctionne indépendamment de l'endroit où les applications sont installées. La permission "*signatureOrSystem*" est utilisée pour certaines situations spéciales où plusieurs fournisseurs ont des applications intégrées dans une image système et doivent partager des

	fonctionnalités spécifiques explicitement parce qu'elles sont construites ensemble.

2.1.3.1 Les permissions d'application avant la mise à jour Android 6.0 Marshmallow

Les permissions sont un aspect du modèle de sécurité d'***Android***, qui permet aux utilisateurs d'avoir un pouvoir sur ce qu'ils peuvent et ne peuvent pas accepter sur leurs appareils mobiles. Comme mentionné précédemment les applications ***Android*** sont exécutées dans un *sandbox* à application individuelle avec une gamme limitée de ressources système. Lorsqu'une application est installée sur un périphérique exécutant Android 5.1 ou inférieur, ***Android*** restreint automatiquement la capacité de l'application jusqu'à ce que l'utilisateur accepte explicitement l'installation et l'accès au système d'exploitation ou aux fonctionnalités requises pour fonctionner correctement. La figure ci-dessous décrit l'affichage habituel des permissions demandées par une application à l'utilisateur au moment de l'installation.

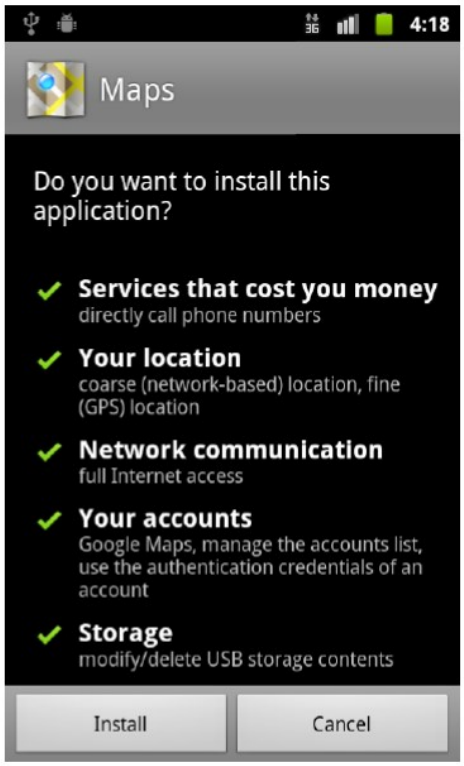

Figure 6 - Affichage des permissions Android requises pour 'Maps' [13]

Sur les périphériques exécutant ***Android* 5.1** ou moins, les permissions d'une application sont toutes requises ou rien : les utilisateurs n'ont pas le choix, ils doivent accepter toutes les permissions d'une application ou refuser l'application complètement et dans ce cas, il ne peut pas utiliser l'application parce qu'il n'est pas d'accord avec certaines permissions. Ce concept empêche les développeurs légitimes de se préoccuper du fait que les utilisateurs puissent ou non refuser des permissions uniques ce qui risque de provoquer une panne de leur application ou un comportement incorrect.

2.1.3.2 Android 6.0 Marshmallow : Les modifications apportées au modèle de permission

Android 6.0 Marshmallow a été publié en octobre 2015. Le but de chaque mise à jour ***Android*** conçue est de donner à l'utilisateur un environnement mobile plus interactif, plus convivial et plus sûr. La mise à jour 6.0 s'est vantée de fournir aux utilisateurs une assistance contextuelle, une meilleure durée de vie de la batterie et des niveaux plus élevés de productivité, de connectivité et d'internationalisation, mais ses plus grandes améliorations ont été les changements dynamiques apportés à la fonctionnalité de sécurité et à la réorganisation du gestionnaire des permissions [19].

À partir de la version d'***Android* 6.0**, les utilisateurs peuvent maintenant accorder des permissions pendant l'exécution des applications, ne demandant plus la permission d'être accordée lors de l'installation initiale d'une application, précédemment abordée dans la section 2.1.3.1

Cette approche est conçue pour simplifier le processus d'installation de l'application, et pour donner à l'utilisateur plus de contrôle. En fait, la mise à jour 6.0 a fourni aux utilisateurs une fonctionnalité avancée et un contrôle sur leurs applications, en leur donnant le pouvoir de révoquer les autorisations individuelles d'une application via l'interface des paramètres de l'application [20]. Par exemple, un utilisateur pourrait choisir de donner à une application de transport un accès à l'emplacement de son appareil mais

rejeter l'accès pour afficher sa liste de contacts ou ses services SMS. La figure ci-dessous montre la nouvelle interface utilisateur pour ce contrôle avancé [20].

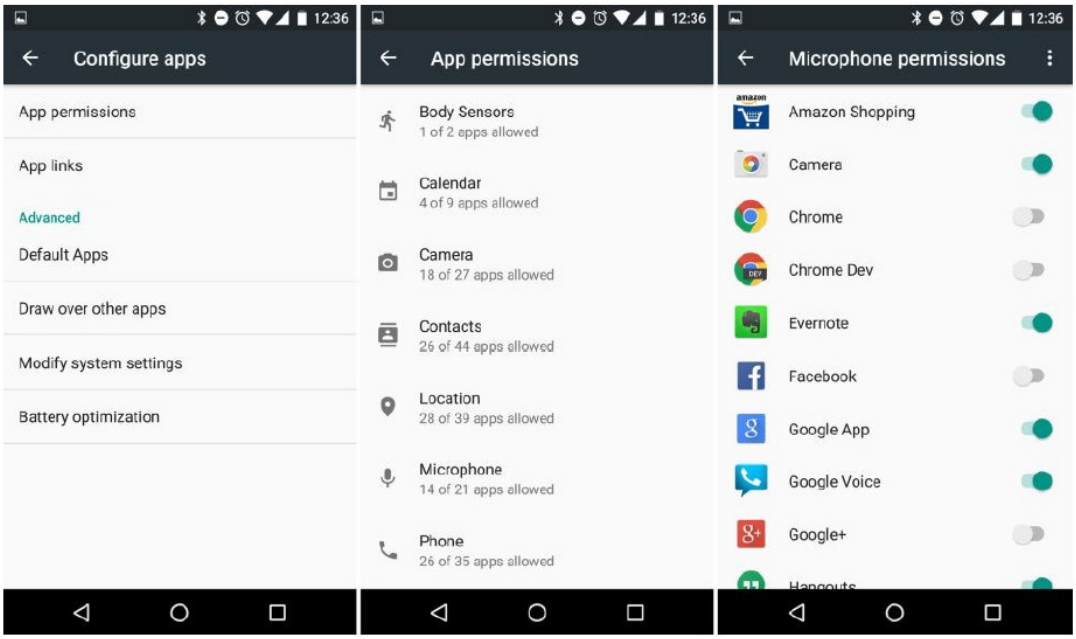

Figure 7 - Paramètres de permission d'une application [20].

Un périphérique exécutant ***Android* 6.0** ou supérieur définit maintenant les permissions dangereuses dans ce qu'***Android*** appelle « groupes de permission ». Le tableau 2 décrit les 9 groupes de permissions et leurs permissions dangereuses associées. Ce tableau a été créé par Google et dans lequel ils classent les permissions dangereuses par groupe de permissions [16]. Le système indique simplement à l'utilisateur les groupes de permissions dont l'application a besoin, et non les permissions individuelles.

Tableau 2 - Groupe de permissions dangereuses

Groupe de permission	**Permission dangereuse**
Calendrier	Lire_Calendrier Ecrire_Calendrier

Camera	Camera
Contacts	Lire_Contact Ecrire_Contact Obtenir_Compte
Location	*ACCESS_FINE_LOCATION* *ACCESS_COARSE_LOCATION*
Microphone	Enregistrement_Audio
Phone	Lire_L'état_de_téléphone Appel_Téléphonique Lire_Journal_D'appels Ecrire_ Journal_D'appels Ajouter_des messages vocaux Utilisation_SIP Processus_Appels_Sortants
Capteurs	Capteurs_Du_Corps
SMS	Envoyer_SMS Recevoir_SMS Ecrire_SMS Recevoir_MMS
Stockage	Lire_Stockage_Externe Ecrire_ Stockage_Externe

2.2 Travaux reliés

Les travaux relatifs à ce mémoire se divisent en quatre catégories soit la propagation d'informations dans les applications mobiles, l'analyse du modèle de permission ***Android***, la détection de malware et autres études empiriques.

2.2.1 Propagation d'informations dans les applications mobiles

La communication inter-application permet aux applications de communiquer avec des services tiers et d'autres applications, afin de créer une interface utilisateur transparente. Chin et al. (2011) ont créé un outil appelé ***'ComDroid'*** pour détecter les vulnérabilités dans les systèmes de communication d'applications. ***ComDroid*** peut être utilisé par les développeurs pour analyser leurs propres applications avant de la publier sur l'***Android Market*** [21]. ***ComDroid*** est un outil d'analyse statique qui permet d'identifier et d'émettre des avertissements, mais il ne peut pas vérifier l'existence d'attaques ou de corrections disponibles pour les vulnérabilités trouvées dans la conception de l'application. ***ComDroid*** ne fournit pas une analyse dynamique et il ne peut pas identifier les vulnérabilités pendant qu'une application est en cours d'exécution, réduisant la portée de la détection de vulnérabilité. Les chercheurs ont réalisé une analyse de 20 applications à l'aide de ***ComDroid*** et ils ont trouvé 34 vulnérabilités exploitables dans 12 des applications. Octeau et al. (2013) ont également développé un outil appelé ***'Epicc'*** (Efficient and Precise ICC). ***« Epicc »*** a été conçu pour identifier les vulnérabilités de communication entre composants (ICC) (autrement connues sous le nom de communication inter-applications) [22]. ***« Epicc »*** comme ***« ComDroid »*** était un outil d'analyse statique, mais il a été conçu avec un modèle d'analyse supérieur, permettant à l'outil de se brancher davantage dans la structure de communication des applications d'***Android***, non seulement en révélant toute vulnérabilité dans une application, mais en stockant les données pour une future analyse d'application. 1200 applications ont été analysées dans leur étude, avec une analyse moyenne prenant 113 secondes par application, ce qui démontre la capacité de l'outil. Comme il ne fournit pas d'analyse dynamique, il partage également la même limite que « ***ComDroid*** ».

Rubin et al (2015) ont proposé une approche pour distinguer entre la communication ouverte qui contribue à la fonctionnalité d'application anticipée par l'utilisateur et la communication secrète cachée et inattendue du point de vue de l'utilisateur [23]. Ils ont analysé les modèles de communication des applications populaires dans le marché *Google Play*. Motivés par la quantité importante de communications secrètes qu'ils ont trouvées dans ces applications, ils ont développé une analyse statique hautement précise et évolutive capable d'identifier automatiquement une telle communication. Leur travail se concentre sur la distinction de la communication secrète dans des applications bénignes telles que Twitter, Spotify, tout en étant caché et inattendu du point de vue de l'utilisateur.

La technique la plus importante pour le suivi dynamique de la propagation de l'information dans ***Android*** est ***TaintDroid***. Enck et al. (2010) ont utilisé cet outil pour étudier le comportement de 30 applications tierces populaires, choisies au hasard dans ***Android Market***. L'étude a révélé que les deux tiers des applications présentent une manipulation suspecte de données sensibles et que 15 des 30 applications ont rapporté les emplacements des utilisateurs aux serveurs de publicité à distance [24].

Ces travaux sont complémentaires à notre approche. L'analyse de code est une tâche complexe et lourde, cependant l'application de l'une de ces techniques de suivi de flux d'informations sur les applications mobiles peut être une perspective de continuité pour notre travail dans le futur.

2.2.2 Analyse du modèle de permission Android

De nombreux chercheurs ont proposé des approches pour aider les utilisateurs ou les développeurs à comprendre les autorisations utilisées dans les applications ***Android***.

Des études antérieures ont été réalisées en combinant le traitement du langage naturel et le développement de logiciels. ***CHABADA*** [25] utilise la modélisation de sujet en utilisant *Latent Dirichlet Allocation* (LDA) pour évaluer la description de l'application contre le comportement de l'application. Il génère des *clusters* en fonction du thème, qui se compose d'un groupe de mots qui se produisent souvent ensemble. Ensuite, il tente de détecter les valeurs aberrantes en tant que comportements malveillants. ***CHABADA*** ne

peut pas identifier les fuites de confidentialité, car il identifie simplement les *API* sensibles sans tracer le flux de données entre les *API* sensibles aux sources et aux puits. Le travail le plus proche à ***CHABADA*** est le ***Framework WHYPER*** de Pandita et al. (2013) [26]. ***WHYPER*** tente d'automatiser l'évaluation des risques des applications ***Android*** et applique le traitement du langage naturel aux descriptions de l'application. Le but de ***WHYPER*** est de déterminer si le besoin de permissions sensibles (comme les accès aux contacts ou au calendrier) est motivé dans la description de l'application.

Frank et al. (2012) ont sélectionné 188 389 applications provenant du marché officiel d'***Android*** et ont analysé les combinaisons de demandes de permissions par ces applications [27]. Une méthode probabiliste a été proposée pour déduire les modèles de permissions les plus répandus en fonction de la popularité des applications (notes et nombre de revues). Les auteurs ont identifié plus de 30 modèles communs de demandes de permissions. Leur objectif principal était d'étudier l'écart des demandes de permissions pour les applications à rang élevé et celles à rang faible, les résultats ont montré que les applications à faible réputation s'écartent souvent des modèles de demande de permissions identifiés pour les applications à haute réputation.

D'une manière similaire à ce de dernier travail, dans la première partie de notre mémoire, nous nous intéressons à l'étude des différentes combinaisons de demandes de permissions. Cependant, nous ne nous restreindrons pas l'étude à la comparaison des modèles de permissions demandés par les applications à haute et faible réputation. Mais plutôt, nous ferons en sorte que l'étude soit généralisable, et cela en effectuant l'analyse d'un échantillon représentatif des différentes applications qui existent et ainsi nous identifierons des modèles de permissions qui incluront toutes les applications sans tenir compte du facteur de popularité de ces dernières.

2.2.3 Détection de malware

Certains chercheurs comme Zhou et Jiang [28] ont utilisé les permissions demandées par les applications en tant que filtre pour identifier les applications potentiellement malveillantes, la détection actuelle utilise une analyse statique pour

comparer les séquences d'appels *API* contre celles des logiciels malveillants connus. D'autres chercheurs ont utilisé les techniques d'apprentissage machine pour détecter les applications malveillantes d'***Android***. ***MUDFLOW*** [29] est un outil qui exploite l'outil d'analyse statique ***FLOWDROID*** [30] pour déterminer ces flux pour toutes les sources Android sensibles. ***MUDFLOW*** met en œuvre plusieurs classificateurs, formés sur le flux de données d'applications bénignes, pour repérer automatiquement les applications ayant des fonctionnalités suspectes. Dans [31], les auteurs ont analysé l'utilisation des permissions dans un ensemble de 1227 d'applications bénignes et 49 familles d'applications ***Android*** malveillantes. Les résultats ont montré que certaines autorisations peuvent être déclarées mais non utilisées par une application. Les auteurs ont statistiquement analysé et comparé les applications malveillantes et bénignes et les permissions utilisées et requises correspondantes. Les résultats ont montré que les permissions ***Android*** les plus fréquemment utilisées sont assez communes aux deux ensembles d'applications, mais certaines permissions ne sont utilisées que par des applications malveillantes ou bénignes. Enfin, les patrons possibles de 2 à 4 autorisations ont été générés automatiquement et la fréquence d'occurrence a été attribuée à chaque motif. Certains patrons uniques pour la détection des logiciels malveillants ont été identifiés.

2.2.4 Étude empirique sur les applications mobiles

Barrera et al. (2010) ont créé une méthodologie conçue pour analyser empiriquement les permissions des applications ***Android***. La recherche empirique est la méthode permettant d'acquérir des connaissances par observation ou expérience directe ou indirecte d'un sujet, dans ce cas pour étudier comment fonctionne le système basé sur les permissions et identifier les forces et les limites de la mise en œuvre actuelle. Les chercheurs ont analysé en utilisant l'algorithme SOM (Self-Organizing Map) un échantillon de 1 100 applications ***Android*** composées des 50 meilleures applications gratuites téléchargées en 2009 dans chacune des 22 catégories de l'***Android Market*** [7]. Les résultats ont montré qu'un petit sous-ensemble des permissions est utilisé très fréquemment lorsqu'un grand sous-ensemble de permissions a été utilisé par très peu d'applications. Ils

ont de même noté que 62% des applications collectées en décembre 2009 utilisent l'autorisation *INTERNET*.

Harman et al. (2017) ont utilisé les techniques du data mining pour extraire les informations disponibles dans le magasin d'applications *BlackBerry* telles que les descriptions d'applications, les prix, les notes et le nombre de votes. Ils se sont concentrés sur les métadonnées des applications pour trouver des modèles tels que la corrélation entre la note d'une application et le nombre des téléchargements d'une application [32]. Les résultats ont révélé qu'il existe de fortes corrélations entre le nombre de votes des utilisateurs et la popularité (le nombre de téléchargements d'applications). Les auteurs ont prouvé qu'il y'a une corrélation légère entre le prix de l'application et le nombre des fonctionnalités revendiquées pour l'application et ils ont également constaté que les applications à prix plus élevés avaient tendance à être moins bien notées par leurs utilisateurs.

2.3 Conclusion

Dans ce chapitre, nous avons présenté le système d'exploitation ***Android*** et ces applications et nous avons fourni un aperçu sur les travaux reliés aux permissions dans les applications mobiles. Les travaux existants requièrent la présence du code source de l'application et nécessitent des algorithmes sophistiqués pour analyser le code ceci limite leur utilisabilité. L'objectif de notre travail est de proposer une alternative plus légère et plus réaliste. Pour cela, nous proposons un modèle prédictif de risque à partir d'attributs qui sont déjà documentés dans le marché *Google Play*. Nous présentons dans le prochain chapitre notre approche d'identification de patrons d'utilisation de permissions.

CHAPITRE 3

ÉTUDE DE DÉPENDANCE ENTRE PERMISSIONS

3.1 Introduction

L'un des mécanismes de sécurité de la plateforme ***Android*** et un système d'avertissement principal d'utilisateurs d'***Android*** est le système de permission. Chaque application a des capacités très limitées par défaut, et doit exiger des permissions pour accéder aux données ou services sensibles. Lors de l'installation d'une application, l'utilisateur est informé des permissions requises par celle-ci. Il doit fournir son accord à cette liste de permissions et ainsi il limite l'accès de l'application durant l'exécution aux ressources autorisées.

Dans l'***Android Market***, les applications sont regroupées en 27 catégories d'applications et 8 catégories de jeux en fonction de leur objectif principal, par exemple « Livres et références », « Photographie » et « Météo ». Les catégories sont définies dans le but de regrouper les applications qui offrent des fonctionnalités similaires. Nous pensons intuitivement que les applications qui offrent les mêmes fonctionnalités devraient utiliser les mêmes permissions et devraient également suivre les mêmes patrons de permissions.

Le but de ce chapitre est d'identifier des patrons de permissions communs et de vérifier jusqu'à quel point ces patrons sont suivis par les applications ***Android***. Ce chapitre est organisé en quatre sections. La première section présente quelques définitions indispensables pour la compréhension de notre approche, ainsi que l'algorithme que nous avons utilisé. Dans la deuxième section, nous décrivons notre approche d'extraction de patrons de permissions et nous présentons une étude de cas dans la troisième section. Nous clôturons ce chapitre par une conclusion.

3.2 Généralité

3.2.1 Les clusters

Un *cluster* est un ensemble d'éléments. Cet ensemble est distinct des autres. Donc chaque élément d'un *cluster* a de fortes ressemblances avec les autres éléments de ce même *cluster*, et doit être différent des éléments des autres *clusters*. Il existe deux types de *clusters* :

- **Les *clusters* durs** qui sont totalement distincts les uns des autres, ainsi un élément d'un *cluster* n'appartient à aucun autre [33].
- **Les *clusters* mous** comportent des éléments dont l'appartenance est pondérée, donc un élément peut appartenir à plusieurs *clusters* [33].

Les deux **propriétés** importantes définissant un *cluster* pertinent sont :

- Sa cohésion interne (que les objets appartenant à ce *cluster* soient les plus proches possible)
- Son isolation externe (que les objets appartenant aux autres *clusters* soient les plus éloignés possible).

Pour observer cela, plusieurs **mesures** sont associées à un *cluster* :

- Sa densité (la masse d'objets par unité volumique),
- Sa variance (le degré de dispersion des objets dans l'espace depuis le centre du *cluster*),
- Sa dimension (typiquement son rayon ou son diamètre),
- Sa forme (hyper sphérique/allongée/concave/convexe…)
- Sa distance (par rapport aux autres *clusters*).

Si on regarde un *cluster,* il forme un ensemble. Et cet ensemble occupe donc un espace. Pour pouvoir mesurer l'appartenance d'un élément à un *cluster* et pouvoir prendre des décisions, il nous faut une fonction de mesure. On utilise beaucoup la distance de *Minkowski* ou la distance euclidienne [33].

3.2.2 Les méthodes de Clustering

On distingue trois grandes familles de *Clustering* :

- **Le Clustering hiérarchique**, dont le but est de former une hiérarchie de *clusters*, de telle sorte que plus on descend dans la hiérarchie, plus les *clusters* sont spécifiques à un certain nombre d'objets considérés comme similaires [33].
- **Le Clustering par partition**, dont le but est de former plusieurs partitions dans l'espace des objets, selon un certain critère. Chaque partition représente alors un *cluster*. Dans cette famille, on trouve les méthodes suivantes :
    - **Le Clustering K-means**, dont le but est d'identifier un certain nombre (K) de points représentatifs des *clusters*, auxquels sont ensuite associés l'ensemble des autres points, selon leur proximité avec les points représentatifs considérés.
    - **Le Clustering basé sur la densité**, dont le but est d'identifier, dans l'espace, les régions homogènes de fortes densités entourées par des régions de faible densité, qui formeront les *clusters*. DENCLUE [35] et DBSCAN [36] sont des exemples d'algorithmes appartenant à cette catégorie.
    - **Le Clustering basé sur l'utilisation de grilles**, dont l'idée est d'utiliser une grille pour diviser l'espace en un ensemble de cellules, ensuite d'identifier les ensembles de cellules denses connectées, qui formeront les *clusters*.
    - **Le Clustering statistique**, qui fait l'hypothèse que les données ont été générées en suivant une certaine loi de distribution (avec une certaine probabilité), le but étant alors de trouver les paramètres (cachés) de cette distribution.
    - **Le Clustering via la théorie des graphes**, qui cherche, dans un graphe qui connecte les différents objets entre eux, les arcs à conserver pour former les *clusters*.
    - **Les Clustering basés sur la recherche stochastique** : algorithmes génétiques, recherche Tabou ou recuit simulé, parcourent l'espace des

partitions possibles selon différentes heuristiques, et sélectionnent la meilleure qu'ils trouvent dans le temps qui leur est imparti

  - **Le Clustering basé sur les réseaux de neurones**, appelés auto-associatifs, qui recherche les poids à attribuer à l'unique couche du réseau, qui correspondent le mieux à l'ensemble des données

- **Le subspace Clustering**, dont le but est de cibler les *clusters* existant dans des sous-espaces de l'espace original.

3.2.3 Notion de Cluster basé sur la densité

Lorsqu'on visualise les regroupements simples de points de la figure 8, il est possible de détecter facilement et sans aucune ambigüité les points qui appartiennent à un *cluster* et ceux qui n'appartiennent à aucun et qui sont donc non significatifs (on parle dans ce cas de bruit). La principale raison qui nous permet de les reconnaître est qu'à l'extérieur des *clusters* la densité des zones de bruit est inférieure à celle de chacun des *clusters*. Dans ce qui suit, nous essayons de formaliser cette notion intuitive de *cluster* et de bruit dans une base de données D de points d'un espace S de k dimensions. Notez que la notion de *cluster* de l'algorithme ***DBSCAN*** s'applique aussi bien dans un espace 2D, 3D euclidien ou aux espaces comportant de nombreuses dimensions. On fixe **Eps** le rayon du voisinage à étudier et **MinPts** le nombre minimum de points qui doivent être contenus dans le voisinage. L'idée clé du *Clustering* basé sur la densité est que pour chaque point d'un *cluster*, ses environs pour un rayon donné **Eps** doit contenir un nombre minimum de points **MinPts**. Ainsi, le cardinal de son voisinage doit dépasser un certain seuil. Cette forme de voisinage est déterminée par le choix d'une fonction de distance de 2 points p et q, noté dist (p, q). Par exemple, pour un espace 2D, il s'agira d'un rectangle en utilisant une distance de *Manhattan*. Cette approche fonctionne quelle que soit la fonction de distance ce qui permet selon une application donnée de choisir une application appropriée. Afin qu'il soit facilement compréhensible, tous nos exemples seront dans un espace 2D utilisant une distance euclidienne.

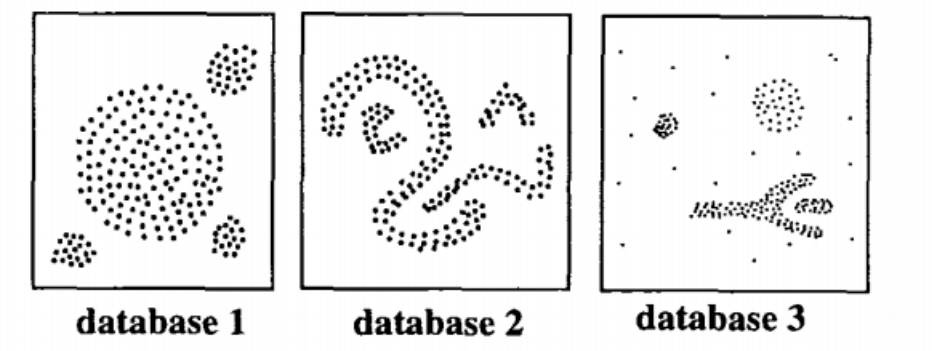

Figure 8 - échantillon de bases de données [36]

3.2.3.1 L'algorithme DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) est un algorithme qui a été conçu pour découvrir les *clusters* et le bruit dans une base de données spatiale [36]. Pour trouver un *cluster,* DBSCAN commence par un point arbitraire p et recherche tous les points de densité accessibles à partir de p. Si p est un point central, la procédure ajoute p au *cluster*. Si p est un point de bordure alors aucun point n'est atteignable à partir de p et DBSCAN visitera le prochain point de la base de données. Grâce à l'utilisation des valeurs globale **Eps** et **MinPts**, *DBSCAN* peut fusionner 2 *clusters* dans le cas où 2 *clusters* de densité différente sont proches l'un de l'autre. Deux ensembles de points ayant au moins la densité la plus petite seront séparés l'un de l'autre si la distance entre les deux est plus large que **Eps**. En conséquence, un appel récursif de *DBSCAN* peut se révéler nécessaire pour les *clusters* détectés avec la plus haute valeur de **MinPts** [36]. Cela n'est pas forcément un désavantage car l'application récursive de *DBSCAN* reste un algorithme basique, et n'est nécessaire que sous certaines conditions. Ainsi, pour chaque objet que l'on ajoute, on a une zone de croissance qui va permettre d'étendre le *cluster*. Évidemment plus cette zone (une sphère) est grande et plus le *cluster* aura de chances de s'étendre. La notion de voisinage est la clé de cette méthode. On forme donc le *cluster* de proche en proche. La difficulté que nous pouvons rencontrer vient de la taille de la zone (rayon de la sphère) d'extension. *DBSCAN* a une complexité en (n log n).

3.2.3.2 Détermination des paramètres Eps et MinPts

Dans cette section, nous allons présenter une **heuristique** simple et efficace afin de déterminer les paramètres **Eps** et **MinPts** du plus petit *cluster* de la base de données. Cette heuristique est basée sur les observations illustrées dans la figure suivante :

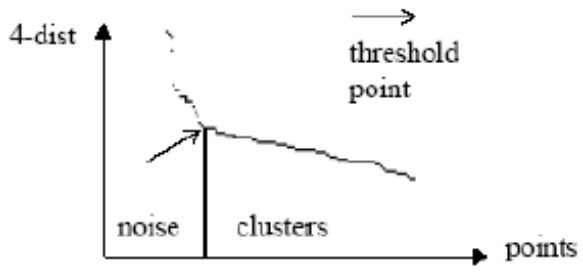

Figure 9 - Heuristique de fixation des paramètres [36]

En général, il peut être délicat de détecter la première vallée automatiquement, mais il est relativement simple pour l'utilisateur de voir cette vallée sur une représentation graphique. C'est pourquoi une approche interactive pour déterminer ce seuil est intéressante. *DBSCAN* a besoin des paramètres **EPS** et **MinPts**. Les expériences ont montré que les graphes de distance k ($k > 4$) ne diffèrent pas vraiment des graphes de distance 4 mais nécessitent des calculs bien plus importants. Ainsi, nous éliminons le paramètre **MinPts** en le fixant à 4 pour toutes les bases de données (d'espace 2D) [36].
Il s'agit ensuite d'utiliser une **approche interactive** pour déterminer le paramètre **Eps** de *DBSCAN* :

- Le système calcule et affiche le graphe de distance 4 pour la base de données.
- Si l'utilisateur peut estimer le pourcentage de bruit, ce pourcentage est entré et le système en déduit une proposition pour le seuil de point.
- L'utilisateur peut accepter ou non le seuil proposé ou sélectionner un autre seuil qui est alors utilisé dans *DBSCAN*.

3.3 Extraction de patrons de permissions

Dans cette section, nous détaillons notre approche pour détecter des patrons d'utilisation de permissions à plusieurs niveaux. Nous fournissons en premier lieu une définition plus approfondie des patrons d'utilisation de permissions à plusieurs niveaux et de leur représentation dans notre approche, ensuite nous décrivons la modification apportée à l'algorithme *DBSCAN* pour nous permettre d'identifier des patrons d'utilisation de permissions.

Nous définissons un Patron d'Utilisation de Permissions (PUP) en tant que groupe (c'est-à-dire, un *cluster*) de permissions que le système ***Android*** utilise pour contrôler les privilèges d'accès à des informations sensibles d'applications. Un Patron d'Utilisation de Permissions (PUP) est un ensemble de permissions que le développeur doit déclarer lors de l'implémentation de l'application. Ces permissions spécifient les ressources sensibles que l'application peut utiliser. Cependant, il est impossible d'analyser tous les scénarios d'utilisations possibles des permissions, en raison de leurs utilisations variées et de leurs grands nombres. Nous avons besoin d'une technique permettant d'identifier les patrons d'utilisation de permissions indépendamment du scénario de déclaration de la permission dans le code source de l'application. Par conséquent, notre technique d'identification des patrons d'utilisation de permissions doit (1) capturer les interférences dans les relations de co-utilisation entre les permissions des applications et (2) isoler les bruits par rapport au degré de relations de co-utilisation dans les patrons détectés.

Notre approche prend en entrée les applications à étudier et les permissions utilisées par chacune des applications. Les résultats de notre approche sont un ensemble de patrons d'utilisation de permissions. L'approche de détection des patrons d'utilisation de permission se déroule comme suit. Dans la première phase, le fichier *manifeste.xml* de l'application est analysé de manière statique pour extraire les permissions utilisées par une application. L'analyse statique est réalisée en utilisant l'outil Androguard [37]. Dans la deuxième phase, nous calculons les vecteurs d'utilisation pour l'application. Chaque application est caractérisée par un vecteur d'utilisation qui code les informations au sujet de sa demande

de permission. Enfin dans la dernière phase, nous utilisons l'analyse de *Clustering* pour regrouper les demandes de permissions les plus souvent co-utilisées par les applications.

Dans notre approche, nous représentons l'ensemble de données des demandes de permissions des applications comme une matrice binaire avec N étant le nombre d'applications, et D le nombre total de permissions possibles x $\in\{0,1\}^{N\times D}$.
L'entrée x_{id} = 1 signifie que l'application i demande la permission d. La ligne $x_{i*} \in \{0,1\}^{D}$ représente toutes les permissions requises de l'application i. Par exemple, dans le tableau 3, nous représentons un vecteur d'utilisation de cinq applications avec sept permissions. Le vecteur d'utilisation de permissions montre que l'application 1 utilise les quatre permissions suivantes: $perm_1$, $perm_2$, $perm_5$, et $perm_6$, on peut remarquer que ces permissions sont également utilisées par l'application 2 et l'application 4.

Tableau 3 - Le vecteur d'utilisation de 5 applications avec 7 permission

App/perm	$perm_1$	$perm_2$	$perm_3$	$perm_4$	$perm_5$	$perm_6$	$perm_7$
app_1	1	1	0	0	1	1	0
app_2	1	1	1	0	1	1	0
app_3	0	1	1	0	1	0	0
app_4	1	1	1	0	1	1	0
app_5	0	0	0	1	0	0	1

DBSCAN construit des groupes de permissions en regroupant ceux qui sont proches les unes des autres et forment une région dense. À cette fin, nous définissons la similarité de permissions dans l'équation (1), entre deux $perm_i$ et $perm_j$, en utilisant le coefficient de similarité de *Jaccard* par rapport aux applications. La raison derrière cela est que deux applications sont proches les unes des autres (courte distance) si elles partagent un grand sous-ensemble de permissions communes.

$$\text{Similarité de permissions } (perm_i, perm_j) = \frac{|\text{App}(perm_i) \cap \text{App}(perm_j)|}{|\text{APP}(perm_i) \cup \text{App}(perm_j)|}$$

Où App ($perm_i$) est l'ensemble de permissions qui utilisent l'application en question. Par exemple, la Permission de Similarité entre l'application 1 et l'application 2 dans le tableau est $\frac{4}{5}$ puisque ces applications ont au total 5 permissions, et 4 d'entre eux sont en communs. La distance entre deux permissions $perm_i$ et $perm_j$ est alors calculée comme suit :

Dist = 1 - Similarité de permissions ($perm_i$, $perm_j$)

DBSCAN a besoin de deux paramètres pour effectuer le *Clustering*. Le premier paramètre est le nombre minimum de permissions dans un *cluster*. Nous l'avons fixé à deux, de sorte qu'un patron d'utilisation doit inclure au moins deux permissions de l'application étudiée. Le deuxième paramètre, epsilon, est la distance maximale à l'intérieur de laquelle deux permissions peuvent être considérées comme voisines l'une de l'autre. En d'autres termes, la valeur epsilon contrôle la densité minimale qu'une région en *cluster* peut avoir. Plus la distance entre les permissions d'un *cluster* est courte, plus le *cluster* est dense. Comme nous le montrerons dans ce qui suit, nous utiliserons différentes valeurs pour epsilon afin d'identifier les patrons d'utilisation à plusieurs niveaux.

Dans *DBSCAN*, la valeur du paramètre epsilon influence énormément les *clusters* résultants. En effet, dans notre approche, une valeur de 0 pour epsilon, signifie que chaque *cluster* doit contenir uniquement des permissions qui sont complètement similaires (c'est-à-dire que la distance entre les permissions appartenant au même *cluster* doit être 0). La relaxation du paramètre epsilon va détendre la contrainte sur la densité demandée dans les *clusters*. D'une part, si nous attribuons à epsilon une petite valeur fixe, comme epsilon = 0, cela produira des patrons d'utilisation très denses. Cependant, les patrons d'utilisation résultants ne captureront pas les interférences dans les relations de co-utilisation entre les permissions de l'application : un patron d'utilisation inclura seulement les permissions qui sont toujours co-utilisées ensemble. D'autre part, en fixant epsilon à une valeur relativement grande, comme epsilon = 0.6, *DBSCAN* produira des patrons d'utilisation qui incluent certains bruits. Pour un patron d'utilisation donné, il ne sera pas facile de distinguer les sous-ensembles denses des sous-ensembles qui capturent l'interférence dans les relations de co-utilisation.

Dans notre approche, nous avons décidé de construire les *clusters* de manière incrémentielle en relaxant le paramètre epsilon, étape par étape. La figure 10 montre le pseudocode du *Clustering* incrémental. L'approche de *Clustering* incrémental a été utilisée dans [38] pour identifier des patrons d'utilisation d'API.

```
HDBSCAN(DataSet, maxEpsilon, MinPts,
    espilonStep){
  epsilon <- 0
  while(epsilon < maxEpsilon)
       DBSCAN(DataSet, epsilon, MinPts)
       clusters <- DBSCAN.clusters
       noisyPoints <- DBSCAN.noisyPoints
       compositePoints <- constructPoints(
           clusters)
       DataSet <- noisyPoints + compositePoints
       epsilon <- epsilon + epsilonStep
}

constructPoints(clusters){
  for each C in clusters
       compositePoint <- OR(all points of C)
}
```

Figure 10 - Algorithme hiérarchique DBSCAN [38]

Au début, nous construisons un ensemble de données contenant toutes les permissions et les regroupons en utilisant l'algorithme *DBSCAN* avec la valeur epsilon de 0. Il en résulte des *clusters* de permissions qui sont toujours utilisées ensemble, et plusieurs permissions bruyantes sont laissées de côté. À la fin de cette exécution, pour chaque *cluster* produit nous regroupons les vecteurs d'utilisation de ses permissions en utilisant la disjonction logique dans un vecteur d'utilisation. Ensuite, un nouveau jeu de données est formé qui inclut les vecteurs d'utilisation agrégés et les vecteurs d'utilisation des permissions bruitées à partir de la première exécution. Cet ensemble de données est renvoyé à l'algorithme *DBSCAN* pour le *Clustering*, mais avec une valeur légèrement supérieure à epsilon, c'est-à-dire epsilon = 0 + δ. Cette procédure est répétée à chaque étape correspondante à une valeur epsilon de α. Et, le processus de *Clustering* est arrêté quand epsilon atteint une valeur maximale, β, donnée en paramètre.

3.4 Étude de cas

3.4.1 Ensemble de données

Nous avons commencé à partir d'un ensemble de données existant [27] exploré depuis le Google Play Store en 2011. Pour l'étude de l'utilisation de la permission, nous avons nettoyé l'ensemble de données pour conserver uniquement les applications qui ont été évaluées par les utilisateurs finaux et qui ont déclaré l'utilisation de l'autorisation. Nous avons terminé avec un total de 138610 applications utilisant 173 permissions différentes. Pour chaque application, nous extrayons les permissions utilisées par cette dernière. Pour ce faire, nous suivons cette méthodologie : en premier lieu, pour lire les fichiers de manifeste, nous décomposons l'*APK* de l'application et nous décodons les ressources en utilisant *Apktool* [39], après avoir obtenu les ressources décompressées pour les applications portables, nous devons identifier le chemin d'accès au fichier *APK* de la version portable, alors nous appliquons les étapes suivantes :

1) extraire le fichier *AndroidManifest.xml* à partir du répertoire principal,

2) analyser l'arbre *XML* du fichier manifeste,

3) sélectionner la balise métadonnée *<uses-permission>* qui se réfère aux permissions, pour récupérer les permissions définies par les développeurs.

3.4.2 Calcul du score de la conformité

Une permission peut être utilisée pour limiter l'accès à des composants ou des fonctionnalités spécifiques de cette application ou d'autres applications. Une application peut demander une ou plusieurs permissions et un patron d'utilisation de permission représente un groupe de permissions qui sont toujours co-utilisées ensemble. Pour calculer le score de conformité d'une application dans une catégorie aux patrons d'utilisation de permissions obtenus suite à l'application notre approche nous avons utilisé la métrique suivante :

La moyenne de score de conformité combiné = $\frac{(moyenne\ de\ score\ de\ conformite + (1 - score\ de\ conformite\ concernant\ la\ permission\ pas\ dans\ les\ patrons))}{2}$

Le score de conformité moyen pour chaque application est le score de conformité moyen pour les patrons admissibles. Le score de conformité pour le patron admissible est la taille de l'intersection (patron et application) sur la taille du patron.

Score de conformité concernant la permission pas dans le patron = $\frac{le\ nombre\ de\ permission\ requise\ dans\ application,\ \ mais\ non\ requise\ \ dans\ l'un\ des\ patrons\ admissibles}{le\ nombre\ total\ de\ permissions\ requises\ dans\ l'APP}$

Nous nous sommes intéressés au calcul du score de conformité d'une application dans une catégorie au patron d'utilisation de permissions pour connaitre si les applications suivent des patrons d'utilisations de permissions et jusqu'à quels points ces patrons suivis sont respectés.

3.4.3 Patron d'utilisation de permissions

Les patrons d'utilisation de permissions ne sont pas disjoints : les permissions peuvent être des membres de plusieurs patrons, et les applications peuvent demander plusieurs patrons. Par conséquent, la plupart des patrons n'incluent qu'un petit nombre de permissions. Un patron d'utilisation de permissions avec une seule permission reflète le fait que la permission est demandée très fréquemment, mais pas toujours avec les mêmes autres permissions. Dans ce qui suit, nous présentons les patrons d'utilisations de permissions de 8 catégories : Business, Communication, Météo, Divertissement, Transportation, Photographie, Shopping et Social.

Patron « Business ». Les permissions les plus centrales du patron que nous avons extrait pour la catégorie « Business » sont *full internet access 88,91*%, *view network state 60,68*%, *read phone state and identity 38,44*%, *modify/delete SD card contents 33,78*%et *add or modify calendar events and send email to guests 32,09*%. Cette dernière permission est utilisée pour permettre aux applications de cette catégorie de gérer les calendriers de leurs utilisateurs et ainsi assurer le bon déroulement à temps de leurs évènements.

Patron « Communication ». Les permissions les plus utilisées dans le patron que nous avons extrait pour la catégorie « Communication » sont *full Internet access 81,13*%, *view network state 60,40*%, *read contact data 51,75*%, *receive SMS 41,67*%, *read SMS or MMS 34,69*%, *edit SMS or MMS 94,22*%. Les applications de cette catégorie utilisent ces permissions probablement pour assurer les différentes fonctionnalités qui permettent à l'utilisateur de communiquer. La permission *receive SMS* permet à l'application de recevoir des messages alors que la permission *read SMS or MMS* permet à l'application de lire de messages ou des MMS.

Patron « Météo ». Les permissions les plus demandées dans le patron que nous avons extrait pour la catégorie « Météo » sont *full internet access 94,22*%, *view network state 60,40*%, *read phone state and identity 47,99*%, *access_fine_location 43,63*% et *access_coarse_location 36,40*%. Les applications de cette catégorie ont besoin de ces permissions pour fournir les prévisions météorologiques.

Patron « Divertissement ». Les permissions les plus collectées dans le patron que nous avons extrait pour la catégorie « Divertissement » sont *full internet access 89,98*%, *view network state 70,43*%, *read phone state and identity 63,93*%, *prevent device from sleeping 58,99*%, *automatically start at boot 43,68*% et *intercept outgoing calls 33,86*%. La permission *read phone state and identity* permet à l'application d'accéder aux numéros d'identification uniques de l'appareil (IMEI, IMSI). La permission *prevent device from sleeping* empêche l'appareil de se mettre en mode veille même lorsqu'il n'est pas utilisé. La permission *automatically start at boot* permet de lancer automatiquement l'application au démarrage du téléphone et la permission *intercept outgoing calls* permet à l'application d'intercepter les appels sortants.

Patron « Transportation ». Dans le patron que nous avons extrait pour la catégorie « Transportation », *access_fine_location 90,26*% et *access_coarse_location 77,40*% sont clairement les permissions les plus importantes pour le transport, puisque la plupart des applications de cette catégorie sont des Maps et des logiciels liés au GPS ainsi que la permission full internet accees.

Patron « Photographie ». La permission de la catégorie « Photographie » la plus centrale dans le patron est la caméra *90,28*% possiblement pour accéder et utiliser une

caméra intégrée à l'appareil pour prendre des photos. Les applications de la catégorie « Photographie » utilisent la permission *set_wallpaper 70,48*% avec la permission caméra pour permettre d'utiliser la photo comme fond d'écran.

Patron « Shopping ». Les permissions les plus déclarées dans le patron que nous avons extrait pour la catégorie « Shopping » sont *full internet access 90,72*%, *view network state 71,93*%, *access_fine_location 51,87*%, *access_coarse_location 42,33*%, *modify/delete USB storage contents 37,63*% et *control vibrator 33,23*%. La permission *modify/delete USB storage contents* permet de modifier ou supprimer le contenu du stockage USB. La permission *control vibrator* permet de contrôler la commande de vibreur de l'appareil.

Patron « Social ». Les permissions les plus déclarées dans le patron que nous avons extrait pour la catégorie « Social » sont *full internet access*, *view network state 90,96*%, *access_fine_location 67,22*%, *access_coarse_location*, *read contact data 52,13*%, *write contact data 43,76*%, *take pictures and videos 39,75*%. La permission *read contact data* permet de lire les données de contact, la permission *write contact data* permet d'ajouter des nouvelles données de contact et la permission *take pictures and videos* permet de prendre des photos et des vidéos.

Nous avons remarqué que la permission *full internet access* a été utilisée par toutes les applications et elle a existé dans tous les patrons d'utilisation de permissions. L'utilisation de la permission *full internet access* indique que la raison derrière la déclaration de cette permission est de permettre à l'application de communiquer avec un serveur distant. De même, nous avons constaté qu'il y'a des permissions communes entre les différents patrons et qu'il y'a une certaine ressemblance entre les patrons d'utilisation de permissions de différentes catégories tels que le patron de la catégorie communication et le patron de la catégorie social. Dans le tableau ci-dessous, nous présentons le nombre de patron de permissions obtenus pour chaque catégorie et le nombre minimal et maximal de permissions dans chaque patron ses différentes catégories.

Tableau 4 – Sommaire des patrons des huit catégories

Catégorie	Nombre de patron	Nombre de permission minimal	Nombre de permission maximal
Business	41	1	4
Communication	41	1	3
Météo	17	1	5
Divertissement	39	1	10
Transportation	30	1	4
Photographie	24	1	4
Shopping	27	1	6
Social	41	1	8

3.4.4 Relation entre la conformité aux patrons et les caractéristiques des applications

Les caractéristiques d'une application dans notre étude de cas représentent les informations disponibles sur le marché *Google Play* telles que le nombre de téléchargements, le nombre de votes, la note moyenne, les applications reliées, le prix et la catégorie.

Dans le tableau ci-dessous, nous représentons les statistiques descriptives de la variable score de conformité aux patrons pour quatre catégories **Finance**, **Météo**, **Communication** et **Outils**.

Tableau 5 – Statistiques descriptives de la variable score de conformité aux patrons de la catégorie Finance, Météo, Communication, Outils

Catégorie	Totale	Minimum	Maximum	Moyenne	Écart type
Finance	2798	0,000	1,000	**,67616**	,238522
Météo	805	0,217	1,000	**,69661**	,155522
Communication	1992	0,000	1,000	**,61273**	,184518
Outils	1735	0,000	1,000	**,58888**	,307101

D'après les résultats de tableau ci-dessus, nous pouvons constater que la moyenne des 3 premières catégories **Finance**, **Météo** et **Communication** est proche alors que la moyenne de la 4e catégorie est un peu plus éloignée de la moyenne des autres catégories. En effet, cette différence est due à la généralité de la catégorie Outils qui peut inclure diverses applications qui ont des fonctionnalités différentes. La **meilleure** catégorie est la catégorie **Météo**, cette catégorie possède la moyenne la plus élevée et l'écart type le plus bas c'est à dire elle représente la catégorie dans laquelle il existe des patrons d'utilisation de permissions et ils sont suivi uniformément par les applications. Nous pouvons expliquer cela par l'homogénéité des applications incluant cette catégorie. La **pire** catégorie est la catégorie **Outils**, elle possède la moyenne la plus basse et l'écart type le plus élevé c'est-à-dire elle représente la catégorie dans laquelle il existe moins de patrons d'utilisation de permissions et ils ne sont pas suivis uniformément par les applications. Si nous comparons la **meilleure** catégorie **Météo** et la **pire** catégorie **Outils,** nous pouvons conclure que le degré de suivi de patrons d'utilisation de permissions diffère d'une catégorie à l'autre. Les applications dans les différentes catégories ne suivent pas toutes de la même façon les patrons d'utilisation de permissions.

Nous avons étudié la relation entre la conformité aux patrons et les caractéristiques des applications. Pour étudier la relation entre la **conformité aux patrons** et les caractéristiques des applications plus précisément la **note moyenne** et le **prix**, nous effectuons un test de normalité (**Kolmogorov-Smirnov**) pour vérifier comment les données (le score de conformité aux patrons, la note moyenne et le prix) sont distribués. Si le score de conformité aux patrons, le prix et la note moyenne sont normalement distribués, nous utilisons la **corrélation de *Pearson*** dans le but d'examiner la nature de la relation entre le **score de conformité**, la **note moyenne** et le **prix**. Sinon, nous utilisons la **corrélation de *Spearman*** pour examiner la nature de cette relation. Les résultats du test de normalité sont affichés dans le tableau 6, les résultats montrent que les données ne sont pas normalement distribuées puisque $p = 0{,}000 < 0{,}05$.

Tableau 6 - Test de normalité de score de conformité, note moyenne et le prix

Tests de normalité

	Kolmogorov-Smirnov	
	Statistique	Signification
Score de conformité	0,127	**0,000**
Note moyenne	0,114	**0,000**
Prix	0,468	**0,000**

Comme le score de conformité aux patrons, le prix et la note moyenne ne sont pas normalement distribués, nous allons donc utiliser la **corrélation de Spearman** pour étudier la corrélation entre le score de conformité, la note moyenne et le prix. Nous avons trouvé une **faible corrélation négative significative** entre le score de conformité aux patrons et la note moyenne et de même une **faible corrélation négative significative** entre le score de conformité aux patrons et le prix. Le tableau ci-dessous représente les résultats de corrélation. Ce résultat est surprenant parce qu'on s'attendait que les applications payantes soient rigoureuses et donc conformes aux patrons. Une explication possible de ce résultat est la suivante : les applications payantes offrent plus de fonctionnalités que les applications gratuites et donc feraient appel à des permissions particulières. Une deuxième explication est que nous avons un débalancement de données dans le sens que nous avons

plus d'applications gratuites que d'applications payantes ce qui a une influence sur les patrons extraits.

Tableau 7 – Corrélation entre le score de conformité, la note moyenne et le prix

Corrélations

			Score de conformité	Note moyenne	Prix
Rho de Spearman	**Score de conformité**	Coefficient de corrélation	1,000	**,026***	**-,096****
		Sig. (bilatérale)	.	,025	,000
		N	7330	7330	7330
	Note moyenne	Coefficient de corrélation	**,026***	1,000	-,009
		Sig. (bilatérale)	,025	.	,466
		N	7330	7330	7330
	Prix	Coefficient de corrélation	**-,096****	-,009	1,000
		Sig. (bilatérale)	,000	,466	.
		N	7330	7330	7330

*. La corrélation est significative au niveau 0,05 (bilatéral).

**. La corrélation est significative au niveau 0,01 (bilatéral).

3.5 Conclusion

Ce chapitre présente une nouvelle technique qui identifie les patrons d'utilisations de permissions. Nous avons présenté les patrons extraits pour 8 catégories. Nous avons trouvé qu'il y'a une faible corrélation négative entre le score de conformité aux patrons, le prix et la note moyenne. Dans le prochain chapitre, nous étudions la relation entre les permissions et le risque.

CHAPITRE 4

RELATION ENTRE PERMISSIONS ET RISQUE

4.1 Introduction

Les permissions sont la méthode d'***Android*** que le développeur utilise pour montrer comment l'application interagit avec le périphérique de l'utilisateur et à quelles informations l'application aura accès (décrit plus en détail dans la section 2.1.3). L'utilisateur comprend qu'en acceptant ces permissions l'application aura accès à différentes zones de son périphérique. Cependant, il peut ne pas comprendre le danger potentiel associé à chaque permission ni comment les combinaisons de permissions peuvent rendre l'utilisateur vulnérable à une multitude de risques.

Notre premier objectif dans ce chapitre est de déterminer si la satisfaction des utilisateurs reflète le risque et la pérennité et de vérifier si l'utilisation de permissions dangereuses est un indicateur de risque et de pérennité. Notre deuxième objectif vise à déterminer si les patrons d'utilisation de permissions peuvent aider à identifier les applications malicieuses pour lesquelles les utilisateurs doivent être prudents lors de l'installation.

Ce chapitre est structuré comme suit. Section 4.2 présente notre ensemble de données. Nous définissons une métrique pour calculer le score de risque d'une application dans la section 4.3. Nous expliquons nos questions de recherche dans la section 4.4 et nous présentons nos résultats dans la section 4.5. Dans la section 4.6, nous discutons plus en détail nos résultats de recherche et nous concluons à la section 4.7.

4.2 Jeux de données

Nous avons implémenté un robot de navigation Web personnalisé pour collecter des données brutes de la page Web du marché d'application *Google Play*. En raison de l'existence d'un grand nombre d'applications, la boutique d'applications *Google Store* ne

fournit pas un moyen direct d'accéder à toutes les applications. Ainsi, notre robot d'exploration recueille les données de l'application en deux étapes. Tout d'abord, il collecte toutes les informations des catégories et analyse chaque page de catégorie pour trouver la liste des *URL* de toutes les applications de chaque catégorie. Il visite ensuite la page Web de chaque application dans chaque catégorie et l'enregistre en tant que donnée d'application brute. Nous avons extrait des informations portant les informations suivantes (Nom, catégorie, icône, description, prix, temps de publication, version, taille, langue, note moyennes des clients et nombre d'évaluations). Cependant, l'analyse de ce travail est axée uniquement sur les attributs catégorie, prix, et note moyenne des clients. Notre ensemble de données comprend un total de 136274 applications et 173 permissions différentes. Dans le tableau ci-dessous, nous affichons un sommaire de données concernant 29 catégories dans *Google Play Store* pour les applications gratuites et payantes.

Tableau 8 – Sommaire de données concernant 29 catégories dans le marché Google Play

Catégorie	**Nombre d'apps gratuites**	**Nombres d'apps payantes**	**Nombre de permissions dangereuses**
Arcade & Action	5040	1648	8835
Books & Reference	3594	2692	6783
Brain & Puzzle	6117	1479	8079
Business	2520	449	8494
Cards & Casino	1261	501	2039
Casual	4403	1227	7246
Comics	838	211	942

Communication	2863	861	14575
Education	3937	1631	7911
Entertainment	12284	3009	26328
Finance	2715	510	5272
Health & Fitness	2524	1143	5316
Lifestyle	6230	1790	14881
Media & Video	2120	640	5244
Medical	804	678	2086
Music & Audio	3408	698	10100
News & Magazines	3677	346	8760
Personalization	4817	7826	6400
Photography	1360	436	3727
Productivity	3634	1329	10826
Racing	665	138	1700
Shopping	1964	213	4246
Social	2728	470	9220
Sports	3958	1466	8257
Tools	9872	2775	24968

Transportation	1275	255	3841
Travel & Local	4749	1625	16530
Weather	690	181	1727
Totale	100047	36227	234332

Nous avons étiqueté la liste des permissions pour distinguer entre Dangereuse, Normal et Signature en fonction du niveau de protection annoncé dans la documentation Android. Nous avons également utilisé différentes balises pour faire la distinction entre les autorisations donnant accès au matériel et les informations sur l'utilisateur.

Comme nous utilisons des données qui ont été collectées en 2011, nous avons vérifié la disponibilité de l'ensemble des applications au cours du projet de mémoire d'où l'idée de considérer deux ensemble d'applications (les applications qui existent toujours dans le marché Google Play et les applications que nous ne pouvions pas télécharger et non disponible sur le marché Google Play).

Dans le chapitre précèdent nous avons utilisé la catégorie Communication, Finance, Météo et Outils. Dans ce chapitre, nous retenons uniquement les deux catégories Communication et Outils puisque le processus d'étiquetage se fait manuellement et il est difficile et long. De même, le processus de vérification de la disponibilité de chaque application prend du temps. Une fois la vérification terminée, nous avons tenté de localiser et télécharger les applications qui ont été supprimés du Google Play Store, dans des référentiels non officiels, dans les forums et sur les sites Web des développeurs.

4.3 Estimation de risque

Dans cette section, nous estimons le risque d'une application en utilisant le scanner de virus **VirusTotal**. *VirusTotal* est un outil disponible en ligne qui agrège de nombreux produits antivirus et moteurs d'analyse en ligne qui permettent d'analyser les applications ***Android*** pour détecter les échantillons de *malware*. Pour effectuer une analyse, nous

soumettons le fichier *apk* à analyser via une page de soumission. Chaque fichier soumis est ensuite analysé par différents produits antivirus afin de détecter si l'application est malveillante ou pas. *VirusTotal* utilise à ce jour 63 produits antivirus. Le résultat des analyses de chaque antivirus est ensuite retourné à l'utilisateur. Nous avons défini une métrique pour calculer le risque d'une application comme suit :

$$\text{Score de risque } (app_i) = \frac{Nombre\ anti{-}virus\ qui\ ont\ detecte\ cette\ app\ \ malware}{Nombre\ totale\ des\ anti{-}virus\ de\ VirusTotal}$$

Nous avons une deuxième variable pour estimer le risque, intitulée variable de pérennité. Il s'agit d'une variable binaire pour laquelle nous attribuons 0 aux applications qui ont été supprimées du marché *Google Play* et 1 aux applications qui existent toujours. Comme nous utilisons des données qui ont été collectées en 2011, nous avons constaté au cours du projet réalisé en 2017 qu'il y a des applications que nous ne pouvions pas les télécharger à partir du Google Play Store.

4.4 Méthodologie d'analyse

Nous allons, dans cette partie, présenter de façon brève certains des concepts utilisés pour mener nos expérimentations. **L'analyse de corrélation** est une technique statistique utilisée pour mesurer le sens et l'intensité de liaisons de nature linéaire entre deux variables. Cette analyse est effectuée en calculant le coefficient de corrélation linéaire r dont les valeurs varient entre -1 et 1. La valeur absolue de r indique la force de la liaison, et son signe le sens de la liaison. La valeur nulle indique l'absence de lien. Si la liaison est forte, une valeur positive de r indique que deux variables varient dans le même sens, et une valeur négative indique qu'elles varient dans le sens opposé. L'analyse de la corrélation nous permettra ainsi de vérifier si deux métriques sont linéairement interdépendantes, le sens et la force de cette liaison.

Un **test statistique** ou **test d'hypothèse** est, en statistiques, une procédure de décision entre deux hypothèses concernant un ou plusieurs échantillons. Il s'agit d'une démarche consistant à rejeter ou à ne pas rejeter une hypothèse statistique,

appelée hypothèse nulle, en fonction d'un échantillon. Notre choix du test est guidé par la question posée et la structure des données. Dans le cas où les observations suivent une loi normale nous choisissons le test de t (test paramétrique) qui est un des tests statistiques les plus puissants. Dans le cas où les données ne suivent pas une loi normale une approche non-paramétrique est préférable. Ainsi, nous choisissons Mann Whitney puisque les échantillons que nous considérons sont aléatoires et indépendants. En effet, Les tests paramétriques sont un peu plus puissants que les tests non paramétriques. En revanche, ils ne peuvent être utilisés que dans des conditions de normalité alors que les tests non paramétriques sont plus robustes et peuvent s'appliquer indépendamment de la distribution et de la taille de l'échantillon.

4.5 Questions de recherche

Pour atteindre nos objectifs mentionnés dans la section d'introduction, nous avons formulé les questions de recherches suivantes :

QR1. Est-ce que la qualité perçue reflète le risque ?

Nous mesurons la qualité perçue d'une application par la note moyenne accordée par un utilisateur à une application. Nous mesurons le risque d'une application en nous basant sur la métrique définie dans la section précédente. Pour répondre à cette question, nous allons créer deux groupes d'applications, un groupe d'application à faible risque (score de risque = 0) et un groupe d'application à risque élevé (score de risque > 0).

En premier lieu, nous effectuons un test de normalité (***Kolmogorov-Smirnov***) sur les données pour vérifier la distribution des données. Si la note moyenne et le score de risque suivent une loi normale, nous utilisons la **corrélation de *Pearson*** pour étudier la corrélation entre la **note moyenne** accordée par les utilisateurs à une application et le **score de risque**. Sinon si la note moyenne et le score de risque ne sont pas normalement distribués, nous utilisons dans ce cas la **corrélation de *Spearman***. Ensuite, en nous basant sur les résultats de test de normalité, nous utilisons soit un ***T-Test*** pour comparer la variation de la note moyenne entre les applications à faible risque (score de risque = 0) et les applications à risque élevé (score de risque > 0) ou un ***U-Test de Mann-Whitney*** pour

comparer la variation de la note moyenne accordée par les utilisateurs à une application entre ces deux groupes.

QR2. Est-ce que la qualité perçue reflète la pérennité ?

Comme précédemment, nous allons utiliser la note moyenne accordée par les utilisateurs à une application et un ensemble de données qui contient les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées.

Nous procédons de la même façon que dans la première question. Nous effectuons, en premier lieu, un test de normalité (***Kolmogorov-Smirnov***) sur les données pour vérifier comment les données sont distribuées. Si la note moyenne est normalement distribuée, nous utilisons un ***T-Test*** pour comparer la variation de la note moyenne entre les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées. Sinon nous utilisons un ***U-test de Mann-Whitney*** pour comparer la variation de la note moyenne entre ces deux groupes.

QR3. Est-ce que le degré d'utilisation de permissions dangereuses est un indicateur de risques ?

Pour répondre à cette question, nous allons utiliser le nombre de permissions dangereuses déclarées par les applications et un ensemble de données composées des applications à faible risque (score de risque = 0) et des applications à risque élevé (score de risque > 0).

Comme pour les deux questions précédentes, nous effectuons, en premier lieu, un test de normalité (***Kolmogorov-Smirnov***) sur les données pour vérifier la distribution de données. Si le nombre de permissions dangereuses et le score de risque sont normalement distribués, nous utilisons la **corrélation de *Pearson*** pour étudier la corrélation entre **le nombre de permissions dangereuses** et **le score de risque**. Sinon nous utilisons dans ce cas la **corrélation de *Spearman***. Ensuite, en nous basant sur les résultats de la distribution de données, nous utilisons soit un ***T-Test*** pour comparer le degré d'utilisation de permissions dangereuses entre les applications à faible risque (score de risque = 0) et les

applications à risque élevé (score de risque > 0) ou un ***U-Test de Mann-Whitney*** pour comparer le degré d'utilisation de permissions entre ces deux groupes.

QR4. Est-ce que le degré d'utilisation de permissions dangereuses est un indicateur de pérennité ?

Nous procédons de la même façon que précédemment, nous utilisons le nombre de permissions dangereuses déclarées par les applications et un ensemble de données qui contient les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées.

Comme précédemment, nous effectuons, en premier lieu, un test de normalité (***Kolmogorov-Smirnov***) sur les données pour vérifier comment les données sont distribuées. Si le nombre de permissions dangereuses suit une loi normale, nous utilisons un ***T-Test*** pour comparer le degré d'utilisation de permissions dangereuses entre les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées. Sinon nous utilisons un ***U-Test de Mann-Whitney*** pour comparer le degré d'utilisation de permissions dangereuses entre ces deux groupes.

QR5. Est-ce que la conformité aux patrons est un indicateur de risque ?

Nous mesurons le score de conformité aux patrons en nous basant sur la métrique calculée dans la section 3.3.2 et un ensemble de données qui contient les applications à faible risque (score de risque = 0) et les applications à risque élevé (score de risque > 0).

Comme pour les questions précédentes, nous effectuons, en premier lieu, un test de normalité (***Kolmogorov-Smirnov***) sur les données pour vérifier la distribution de données. Si le score de conformité et le score de risque sont normalement distribués, nous utilisons la **corrélation de *Pearson*** pour étudier la corrélation entre **le score de conformité aux patrons** et **le score de risque**. Sinon nous utilisons dans ce cas la **corrélation de *Spearman***.

QR6. Est-ce que la conformité aux patrons est un indicateur de pérennité ?

Nous procédons de la même façon que précédemment, nous utilisons le score de conformité aux patrons calculé précédemment et un ensemble de données qui contient les

applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées.

Nous effectuons comme précédemment, un test de normalité (***Kolmogorov-Smirnov***) sur les données pour vérifier comment les données sont distribuées. Si le score de conformité suit une loi normale, nous utilisons un ***T-Test*** pour comparer le score de conformité aux patrons entre les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées. Sinon nous utilisons ***U-Test de Mann-Whitney*** pour comparer le score de conformité aux patrons entre ces deux groupes.

4.6 Résultats

QR1. Est-ce que la qualité perçue reflète le risque ?

D'après les résultats obtenus dans le tableau 9, nous pouvons conclure que les données ne sont pas normalement distribuées puisque $p = 0{,}000 < 0{,}05$.

Tableau 9 - Tests de normalité de la note moyenne et du score de risque

	Kolmogorov-Smirnov	
	Statistique	Signification
Note moyenne	,107	**,000**
Score de risque	,273	**,000**

Comme la note moyenne et le score de risque ne sont pas normalement distribués, nous allons appliquer la **corrélation de *Spearman*** pour étudier la corrélation entre le **score de risque** et la **note moyenne**. D'après les résultats obtenus dans le tableau ci-dessous, nous pouvons conclure qu'il y'a une **faible corrélation négative** entre le score de risque et la note moyenne, c'est-à-dire lorsque la note moyenne est élevée le score de risque est bas. La corrélation est très faible mais elle est significative. Nous nous attendions à ce résultat, en effet l'utilisateur attribue une bonne note à l'application qui satisfait son besoin et qui ne présente pas un risque élevé sur son périphérique.

Tableau 10 - Corrélation ente la note moyenne et le score de risque

Corrélations

			Note moyenne	Score de risque
Rho de Spearman	**Note moyenne**	Coefficient de corrélation	1,000	**-,060****
		Sig. (bilatérale)	.	,000
		N	3727	3727
	Score de risque	Coefficient de corrélation	-,060**	1,000
		Sig. (bilatérale)	,000	.
		N	3727	3727

**. La corrélation est significative au niveau 0,01 (bilatéral).

Nous avons réalisé un ***U-Test de Mann-Whitney*** pour comparer la variance de la note moyenne entre les applications à faible risque (score de risque = 0) et les applications à risque élevé (score de risque > 0). La variable **Malware Détecté** représente notre critère de regroupement. Elle s'agit d'une variable binaire pour laquelle nous avons attribué 0 aux applications à faible risque (score de risque = 0) et 1 aux applications à risque élevé. D'après les résultats obtenus dans le tableau 11, nous pouvons conclure que les applications à faible risque varient d'une façon significative des applications à risque élevé. En effet, les applications à faible risque ont une note en moyenne plus élevée que celle des applications à risque élevé. C'est à dire les applications à risque élevé sont moins notées que les applications à faible risque. Ainsi, on peut déduire que plus la note moyenne d'une application est basse plus elle aura de chance d'être malveillante. D'après les résultats du test de corrélation et les résultats du test ***Mann-Whitney***, on peut dire que la note moyenne d'une application est un indicateur de risque.

Tableau 11 - U-test de Mann-Whitney pour comparer la note moyenne entre les applications bienveillantes et les applications malveillantes

Statistiques de groupe

	Malware détecté	Totale	Moyenne	Écart-type	Erreur standard moyenne
Note	0	1506	4,017	,6892	,0178
	1	2221	3,926	,7727	,0164

Test[a]

	Note
U de Mann-Whitney	1565750,000
W de Wilcoxon	4033281,000
Z	-3,314
Signification asymptotique (bilatérale)	**,001**

a. Critère de regroupement : Malware Détecté

QR2. Est-ce que la qualité perçue reflète la pérennité ?

Nous avons réalisé dans la question précédente un test de normalité. Nous savons que la note moyenne n'est pas normalement distribuée, c'est pour cette raison nous choisissons de faire un ***U-Test de Mann-Whitney*** pour comparer la note moyenne entre les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées. Les résultats que nous avons obtenus suite à l'application ***d'U-test de Mann-Whitney*** sont présentés dans le tableau 12. Nous avons attribué 0 aux applications qui ont été supprimées du marché *Google Play* et 1 aux applications qui existent toujours. Nous avons remarqué qu'il y'a une différence de moyenne significative entre les applications qui existent toujours dans le marché *Google Play* et les applications qui n'existent plus. En effet, la note est en moyenne plus élevée pour les applications qui existent toujours dans le marché *Google Play* que pour les applications qui n'existent plus. Ainsi on peut conclure que la note moyenne est un indicateur de pérennité, plus la note

moyenne d'une application est basse plus elle aura de chance d'être supprimée. Ce résultat confirme que *Google* prend en considération l'évaluation des utilisateurs des applications et se base sur ce critère pour supprimer les applications inutiles.

Tableau 12- U-test de Mann-Whitney pour comparer la pérennité entre les applications bien notées et les applications les moins notées

Statistiques de groupe

	Existe	Totale	Moyenne	Écart-type	Erreur standard moyenne
Note	0	1833	3,900	,7784	,0182
	1	1894	4,024	,6984	,0160

Test[a]

	Note
U de Mann-Whitney	1565627,500
W de Wilcoxon	3246488,500
Z	-5,191
Signification asymptotique (bilatérale)	,000

a. Critère de regroupement : Existe

QR3. Est-ce que le degré d'utilisation de permissions dangereuses est un indicateur de risques ?

D'après les résultats obtenus dans le tableau 13, nous pouvons conclure que les données ne sont pas normalement distribuées puisque $p = 0,000 < 0,05$.

Tableau 13 - Tests de normalité du nombre de permissions dangereuses et du score de risque

	Kolmogorov-Smirnov	
	Statistique	Signification
Nombre de permissions dangereuses	,196	**,000**
Score de risque	,273	**,000**

Par conséquent, nous allons appliquer la **corrélation de *Spearman***. D'après les résultats obtenus dans le tableau 14, nous pouvons conclure qu'il y'a une **faible corrélation positive** entre le score de risque et le nombre de permissions dangereuses. Cette corrélation est faible mais elle est significative. C'est-à-dire lorsque le score de risque est élevé, le nombre de permissions dangereuses est élevé également. Ce résultat confirme notre intuition que le risque d'une application est lié à la dangerosité des permissions requises. En effet, une application qui demande plus de permissions dangereuses est plus risquée que celle qui demande moins.

Tableau 14 - Corrélation de Spearman entre le score de risque et le nombre de permissions dangereuses

			Score de risque	**Nombre de permissions dangereuses**
Rho de Spearman	**Score de risque**	Coefficient de corrélation	1,000	**,117****
		Sig. (bilatérale)		,000
		N	3727	3727
	Nombre de permissions dangereuses	Coefficient de corrélation	**,117****	1,000
		Sig. (bilatérale)	,000	
		N	3727	3727

Comme les données ne sont pas normalement distribuées, alors nous choisissons de réaliser un ***U-Test de Mann-Whitney*** pour comparer le degré d'utilisation de permissions dangereuses entre les applications à faible risque (score de risque = 0) et les applications à

risque élevé (score de risque > 0). Les résultats que nous avons obtenus suite à l'application ***d'U-test de Mann-Whitney*** sont présentés dans le tableau 15. **Malware détecté** est une variable binaire pour laquelle nous avons attribué 0 aux applications à faible risque et 1 aux applications à risque élevé. D'après ces résultats nous pouvons conclure qu'il y'a une différence significative entre les applications à faible risque et les applications à risque élevé. En effet, les applications à risque élevé demandent en moyenne plus de permissions dangereuses que les applications à faible risque.

Tableau 15 - U-test de Mann-Whitney pour comparer le degré d'utilisation de permissions dangereuses entre les applications bienveillantes et les applications malveillantes

Statistiques de groupe

	Malware Détecté	Totale	Moyenne	Écart-type	Erreur standard moyenne
Nombre de permissions dangereuses	0	1506	2,93	3,370	,087
	1	2221	3,63	4,049	,086

Test[a]

	Nombre de permissions dangereuses
U de Mann-Whitney	1512814,000
W de Wilcoxon	2647585,000
Z	-5,006
Signification asymptotique (bilatérale)	**,000**

a. Critère de regroupement : Malware Détecté

QR4. Est-ce que le degré d'utilisation de permissions dangereuses est un indicateur de pérennité ?

Puisque le nombre de permissions dangereuses n'est pas normalement distribué, nous choisissons alors de réaliser un ***U-Test de Mann-Whitney*** pour comparer le score de risque entre les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées. Les résultats sont présentés dans le tableau 16. Nous avons attribué 0 aux applications qui ont été supprimées et 1 aux applications qui existent toujours. Nous avons remarqué qu'il y a une grande différence de moyenne entre les applications qui existent et les applications qui n'existent plus. En effet, le score de risque est plus élevé pour les applications qui n'existent plus. Ainsi on peut conclure que le score de risque est un indicateur de pérennité. Ce résultat confirme que *Google* applique des analyses d'antivirus régulièrement sur les applications et qu'il supprime les applications malveillantes du store.

Tableau 16 - U-test de Mann-Whitney pour comparer la pérennité entre les applications qui existent toujours dans le marché Google Play et celles qui ont été supprimées

Statistiques de groupe

	Existe	Totale	Moyenne	Écart-type	Erreur standard moyenne
Score de risque	0	1833	,21667	,075308	,001759
	1	1894	,03176	,079596	,001829

Test[a]

	Score de risque
U de Mann-Whitney	388080,500
W de Wilcoxon	2182645,500
Z	-42,519
Signification asymptotique (bilatérale)	**,000**

a. Critère de regroupement : Existe

Ensuite, pour comparer le degré d'utilisation de permissions dangereuses entre les applications qui existent toujours dans le marché Google Play et les applications qui ont été supprimées. Nous avons effectué un ***U-Test de Mann-Whitney***. Nous avons obtenu les résultats présentés dans le tableau 17, à partir de ces résultats nous avons pu remarquer qu'il y a une différence significative entre les applications qui existent toujours dans le marché Google Play et les applications qui ont été supprimées. En effet, les applications qui ont été supprimées demandent en moyenne plus de permissions dangereuses que les applications qui existent toujours dans le marché Google Play.

Tableau 17 - U-test de Mann-Whitney pour comparer le degré d'utilisation de permissions dangereuses entre les applications qui existent toujours dans le marché Google Play et celles qui ont été supprimées

Statistiques de groupe

	Existe	Totale	Moyenne	Écart-type	Erreur standard moyenne
Nombre de permissions dangereuses	0	1833	3,67	4,097	,096
	1	1894	3,04	3,472	,080

Test[a]

	Nombre de permissions dangereuses
U de Mann-Whitney	1592680,500
W de Wilcoxon	3387245,500
Z	-4,408
Signification asymptotique (bilatérale)	**,000**

a. Critère de regroupement : Existe

QR5. Est-ce que la conformité aux patrons est un indicateur de risque ?

D'après les résultats obtenus dans le tableau 18, nous pouvons conclure que les données ne sont pas normalement distribuées puisque $p = 0,000 < 0,05$.

Tableau 18 - Tests de normalité de Score de conformité aux patrons et de Score de risque

	Kolmogorov-Smirnov		
	Statistique	Totale	Signification
Score de conformité aux patrons	,145	3727	**,000**
Score de risque	,273	3727	**,000**

Comme le score de conformité aux patrons et le score de risque ne sont pas normalement distribués, nous allons appliquer la **corrélation de *Spearman*** pour étudier la corrélation entre le **score de conformité aux patrons** et le **score de risque**. D'après les résultats obtenus dans le tableau 19, nous pouvons conclure qu'il y'a une **faible corrélation négative** entre le score de conformité et le risque. Cette corrélation est faible et non significative. Ainsi nous pouvons affirmer que le score de conformité aux patrons n'est pas un indicateur de risque.

Tableau 19 - Corrélation de Spearman entre le score de conformité aux patrons et le score de risque

Corrélations

			Score de conformité aux patrons	Score de risque
Rho de Spearman	**Score de conformité aux patrons**	Coefficient de corrélation	1,000	**-,022**
		Sig. (bilatérale)	.	,171
		N	3727	3727
	Score de risque	Coefficient de corrélation	**-,022**	1,000
		Sig. (bilatérale)	,171	.
		N	3727	3727

QR6. Est-ce que la conformité aux patrons est un indicateur de pérennité ?

Puisque le score de conformité aux patrons ne suit pas une loi normale, nous choisissons de faire un ***U-Test de Mann-Whitney*** pour comparer la variation du score de conformité aux patrons entre les applications qui existent toujours dans le marché *Google Play* et les applications qui ont été supprimées. Les résultats sont présentés dans le tableau 20. D'après ces résultats, nous pouvons conclure qu'il y'a une différence non significative (0,278 > 0,05) entre les deux groupes. En effet, les applications qui existent dans le marché *Google Play* sont en moyenne plus conformes aux patrons que les applications qui ont été supprimées, mais cette différence n'est pas significative donc on ne peut dire que le score de conformité est un indicateur de pérennité.

Tableau 20 - U-test de Mann-Whitney pour comparer la pérennité entre les applications qui existent toujours dans le marché Google Play et celles qui ont été supprimées

Statistiques de groupe

	Existe	Totale	Moyenne	Écart-type	Erreur standard moyenne
Score de conformité aux patrons	0	1833	,59293	,235412	,005499
	1	1894	,61062	,262089	,006022

Test[a]

	Score de conformité aux patrons
U de Mann-Whitney	1700305,000
W de Wilcoxon	3494870,000
Z	-1,086
Signification asymptotique (bilatérale)	**,278**

a. Critère de regroupement : Existe

4.7 Discussion

Dans cette section, nous allons discuter nos résultats concernant l'utilisation des permissions dangereuses. En effet, les 174 permissions uniques échantillonnées à partir des applications ont été divisées en deux catégories différentes : les permissions qui contrôlent le matériel du périphérique et les permissions qui accèdent aux informations de l'utilisateur.

- **Les permissions qui contrôlent le matériel du périphérique** : 100 des 174 permissions uniques collectées à partir des applications échantillonnées (57,8% de toutes les permissions uniques), permettent à une application d'interagir avec les composants matériels d'un appareil ***Android*** et n'ont pas directement accès aux informations de l'utilisateur. Les deux permissions les plus courantes qui prennent le contrôle du matériel de l'appareil sont les permissions qui aident l'utilisateur à se connecter à Internet et à maintenir une connexion constante avec les propriétés du réseau des périphériques. La permission *Full Internet Access* a été utilisée par 72,1% de toutes les applications échantillonnées et la permission *Access network state* a été utilisée par 44,5% des applications de l'ensemble de données. Le tableau 21 présente 10 des 100 permissions uniques qui ont le risque le plus élevé de contrôle du matériel de l'appareil.

Tableau 21 - Permissions qui contrôlent le matériel du périphérique

Permission	Usage/description	Pourcentage
Full_Internet_Access	Permet à une application de se connecter à Internet.	72,1%
Access_Network_State	Permet aux applications d'accéder à des informations sur les réseaux.	44,5%
Storage_Modify/delete_SD Card_Contents	Permet à une application de modifier ou de supprimer du contenu de la carte SD.	30,6%

Read_Phone_ State_And_ Identity	Permet un accès en lecture seule à l'état du téléphone, incluant le numéro de téléphone de l'appareil, les informations actuelles du réseau cellulaire, l'état de tous les appels en cours et une liste des *PhoneAccounts* enregistrés sur l'appareil.	26,8%
Control_Vibrator	Permet le contrôle de vibreur du téléphone.	17,4%
Prevent_Device_From_ Sleeping	Empêche l'appareil de se mettre en mode veille même lorsqu'il n'est pas utilisé.	14,2%
View_Wi-Fi_State	Permet aux applications d'accéder à des informations sur les réseaux Wi-Fi.	8,4%
Automatically_Start_At_ Boot	Permet de lancer automatiquement l'application au démarrage du téléphone.	7,5%
Take_Pictures_And_Videos	Permet à une application de prendre des photos et des vidéos.	6,1%
Record_Audio	Permet à une application d'enregistrer de l'audio.	2,9%

Toutes ces permissions ne sont pas complètement sécurisées lors de l'interaction avec le périphérique d'un utilisateur. En cas d'utilisation incorrecte ou malveillante, une application peut potentiellement endommager le matériel d'un utilisateur, mais ces permissions ne peuvent finalement pas accéder à ses informations.

- **Les permissions qui accèdent aux informations utilisateur** : 74 des 174 permissions uniques collectées à partir des applications échantillonnées (42,2% de toutes les permissions uniques), permettent à une application d'accéder et

d'interagir avec les informations de l'utilisateur sur un appareil ***Android***. Les deux permissions les plus courantes qui peuvent accéder aux informations utilisateur sont les permissions qui déterminent l'emplacement de l'utilisateur, qu'il s'agisse d'un emplacement approximatif ou d'un emplacement précis. La permission *ACCESS_FINE_LOCATION* a été utilisée par 18,4% de toutes les applications de l'ensemble de données et la permission *ACESS_COARSE_LOCATION* a été utilisée par 17,7% de toutes les applications échantillonnées. Le tableau 22 présente 10 des 73 permissions uniques, qui ont le risque le plus élevé lors de l'accès aux informations de l'utilisateur et leurs descriptions associées.

Tableau 22 - Permissions qui accèdent aux informations utilisateur

Permission	**Usage/description**	**Pourcentage**
ACCESS_FINE_LOCATION	Permet à une application d'accéder à un emplacement précis.	18,4%
ACCESS_COARSE_LOCATION	Permet à une application d'accéder à l'emplacement approximatif.	17,7%
Directly_Call_Phone_Numbers	Permet à une application de lancer un appel téléphonique sans passer par l'interface utilisateur pour que l'utilisateur puisse confirmer l'appel.	6,7%
Read_Contact_Data	Permet à une application de lire les données de contacts de l'utilisateur.	6,6%
Send_SMS_Messages	Permet à une application d'envoyer des messages SMS.	2,9%

Write_Contact_Data	Permet à une application d'écrire les données de contacts de l'utilisateur.	2,6%
Read_Call_LOG	Permet à une application de lire le journal des appels de l'utilisateur.	1,2%
Read_Browser_History_And_Bookmarks	Permet à une application de lire l'historique et les favoris du navigateur.	1,2%
Read_Calendar_Events	Permet à une application de lire les données de calendrier de l'utilisateur.	1,1%
Write_Browser_History_And_Bookmarks	Permet à une application d'écrire l'historique et les favoris du navigateur.	1%

La combinaison des permissions qui contrôlent le matériel du périphérique et des permissions qui accèdent aux informations utilisateur rend ce dernier plus vulnérable aux risques des applications malicieuses. Dans ce cadre, nous citons un exemple d'applications malicieuses qui combinent entre les permissions des deux catégories.

L'application **acneurope.android** est une application de la catégorie **Communication** qui a été supprimée du marché *Google Play* et a été analysée par l'outil **Virus total**. Les permissions dangereuses qu'elle utilise sont *DISABLE_KEYGUARD, PROCESS_OUTGOING_CALLS, FULL_INTERNET_ACCESS, WRITE_CALL_LOG,* READ_CALL_LOG, WRITE_EXTERNAL_STORAGE, *READ_PHONE_STATE, CHANGE_WIFI_STATE,* et *READ_CONTACT.*

La permission ***DISABLE_KEYGUARD*** permet aux applications de désactiver le verrouillage du clavier s'il n'est pas sécurisé.

La permission ***PROCESS_OUTGOING_CALLS*** permet à une application de voir le numéro composé pendant un appel sortant avec l'option de rediriger l'appel vers un autre numéro ou d'abandonner complètement l'appel.
La permission ***FULL_INTERNET_ACCESS*** permet aux applications d'ouvrir des sockets réseau.
La permission ***WRITE_CALL_LOG*** permet à une application d'écrire les données du journal des appels de l'utilisateur.
La permission READ_***CALL_LOG*** permet à une application de lire le journal des appels de l'utilisateur.
La permission ***WRITE_EXTERNAL_STORAGE*** permet à une application de lire depuis un stockage externe.
La permission ***READ_PHONE_STATE*** permet un accès en lecture seule à l'état du téléphone, incluant le numéro de téléphone de l'appareil, les informations actuelles du réseau cellulaire, l'état de tous les appels en cours et une liste des *PhoneAccounts* enregistrés sur l'appareil.
La permission ***CHANGE_WIFI_STATE*** permet aux applications de modifier l'état de la connectivité Wi-Fi.
La permission ***READ_CONTACT*** permet à une application de lire les données de contacts de l'utilisateur.

4.8 Menaces à la validité

Cette section identifie et discute les menaces à la validité des résultats et des conclusions faites dans ce mémoire.

4.8.1 Validité de conclusion

La validité de la conclusion statistique fait référence à la mesure dans laquelle l'analyse permet de prendre la bonne décision concernant la vérité ou la vérité approximative de l'hypothèse nulle. Dans notre étude, deux erreurs sont possibles. Ils comprennent :

- **Violation d'hypothèses sur les tests statistiques** : si le test statistique est mal choisi il y a un grand risque de conclusions erronées. Pour s'assurer que les tests sont bons, nous avons vérifié la distribution des données et nous avons utilisé les bons tests en fonction de la normalité. Nous avons sélectionné des tests qui sont robustes aux violations.
- **Pêche et taux d'erreurs** : Nous avons réalisé plusieurs tests mais pas de correction de la signification. Cependant, la plupart des tests ont une valeur-p <0.001. Même avec une correction, elles resteront significatives.

4.8.2 Validité interne

L'une des principales limites de la conception de ce projet est les données sur lesquelles nous avons basées notre analyse. Nous utilisons des données recueillies par d'autres chercheurs dont on ignore leurs processus de collecte de données et la façon avec laquelle ils ont procédé pour obtenir les données. Pour contourner cette menace, nous avons nettoyé les données et nous avons sélectionné des échantillons de données de différentes tailles et de multiples catégories. De même, nous avons mis à jour les informations reliées à ces catégories.

4.8.3 Validité de construction

Les menaces pour construire la validité concernent la relation entre la théorie et l'observation, et concernent principalement, dans notre étude, les erreurs possibles dans nos mesures. Ils comprennent :

- **Mauvaise définition des éléments :** Le mesure de score de risque dépend de l'outil Virus total. Outil qui est quand même fiable et qui qui agrège 63 produits d'antivirus. Par contre le calcul de la pérennité peut être biaisé.
- **Biais mono-opération** : Nous avons utilisé deux variables pour estimer le risque soient le score de risque et la pérennité et les résultats allaient dans le même sens.

- **Confusion entre élément et niveau d'élément** : Parfois le niveau est meilleur discriminant que l'absence/présence d'un élément. Pour minimiser cette menace, nous avons utilisé les corrélations et les tests de groupes.

4.8.4 Validité externe

Dans ce travail, nous avons mené des expérimentations empiriques sur les applications Android. Par conséquent, toutes les conclusions qui découlent de nos résultats restent limités. Nous ne pourrons pas les généraliser pour les applications de d'autres systèmes d'exploitation mobiles.

4.9 Conclusion

Dans ce chapitre, nous avons présenté notre ensemble de données et avons étudié la relation entre la satisfaction des utilisateurs et le risque. Nous avons trouvé que la satisfaction des utilisateurs reflète bien l'existence de risque et de pérennité.

De même, nous avons démontré que le degré de permissions dangereuses est à la fois un indicateur de risque et de pérennité. Cependant, le score de conformité n'est pas un indicateur de risque et n'est pas non plus un indicateur de pérennité. Nos résultats de recherche sont bénéfiques pour les utilisateurs et les développeurs. En effet, notre étude permet à l'utilisateur de se renseigner par rapport aux risques liés aux permissions. Elle lui permet de comprendre les permissions et de consacrer plus d'attention à ces dernières pour protéger son périphérique et ses informations personnelles. Quant aux développeurs, elle leur permet de simplifier le modèle de permission d'***Android*** afin d'utiliser le nombre minimum de permissions et d'assurer quand même le bon fonctionnement des applications.

CHAPITRE 5

CONCLUSION

Nous avons proposé à travers ce mémoire une approche qui permet d'identifier des patrons d'utilisation de permissions. L'approche décrite est basée sur l'algorithme de *Clustering DBSCAN*. Il s'agit d'un algorithme basé sur la densité, c'est-à-dire les clusters sont formés en reconnaissant des régions denses de points dans l'espace de recherche. L'idée principale est que chaque point à regrouper doit avoir au moins un nombre minimum de points dans son voisinage. En d'autres termes, l'algorithme regroupe uniquement les points pertinents et omet les points bruyants.

Dans notre approche, nous représentons l'ensemble des données des demandes de permissions des applications par une matrice binaire telle que N soit le nombre d'applications, et D soit le nombre total de permissions possibles $x \in \{0,1\}^{N \times D}$. Les résultats de notre approche sont un ensemble de patrons d'utilisation de permissions.

L'approche de détection des patrons d'utilisation de permission se compose de trois phases. Durant la première phase, le fichier *manifeste.xml* de l'application est analysé de manière statique pour extraire les permissions utilisées par une application. Dans la deuxième phase, nous calculons les vecteurs d'utilisation pour l'application. Chaque application est caractérisée par un vecteur d'utilisation qui code les informations au sujet de sa demande de permission. Enfin dans la dernière phase, nous utilisons l'analyse de *Clustering* pour regrouper les demandes de permissions les plus souvent co-utilisées par les applications.

Nous avons réalisé une étude de cas sur plus de 170 permissions pour plus de 130 000 applications ***d'Android Market*** qui ont été collectées et réparties selon les catégories. Notre étude de cas avait pour but d'identifier des patrons d'utilisation de permissions pour chaque catégorie et de vérifier jusqu'à quel point les patrons identifiés sont suivis par les applications. Nous avons calculé pour chaque application un score de conformité aux

patrons. Notre étude de cas a démontré que le respect des patrons d'utilisations de permissions diffère d'une catégorie à l'autre. De plus, nous avons étudié la relation entre la conformité aux patrons et les caractéristiques d'une application et avons trouvé qu'il y'avait une faible corrélation négative entre le prix et le score de conformité. En effet, les applications gratuites ont tendance à suivre les patrons d'utilisations de permissions plus que les applications payantes. En outre, nous avons trouvé une faible corrélation négative entre le score de conformité aux patrons et la note moyenne accordée par les utilisateurs à une application.

Nous avons également effectué une étude empirique des permissions dans les applications ***Android***. Cette étude avait pour but d'étudier la relation entre les permissions et le risque. L'étude a démontré que le score de conformité n'est pas un indicateur de risque et pas non plus un indicateur de pérennité. Par contre le degré d'utilisation de permissions dangereuses est à la fois un indicateur de risque et un indicateur de pérennité, en effet, les applications qui demandent plus de permissions dangereuses ont plus de chance d'être des applications malicieuses et d'être supprimées du marché *Google Play*. De même, nous avons trouvé que la satisfaction des utilisateurs reflète à la fois l'existence de risque et de pérennité. En effet, les applications moins notées ont plus de chance d'être des applications malicieuses et d'être supprimées du marché *Google Play*.

Notre étude est bénéfique pour les utilisateurs et les développeurs puisqu'elle permet à l'utilisateur de se renseigner par rapport aux risques liés aux permissions. Elle lui permet également de comprendre les permissions et de consacrer plus d'attention à ces dernières pour protéger son périphérique et ses informations personnelles. Quant aux développeurs, elle leur permet de simplifier le modèle de permission d'***Android*** afin d'utiliser le nombre minimum de permissions et d'assurer quand même le bon fonctionnement des applications.

Malgré ces résultats très encourageants, il existe de nombreuses pistes d'amélioration qui méritent d'être explorées. Dans de futurs travaux, nous comptons utiliser une technique de suivi de flux d'informations dynamique ou statistique dont le but est de visualiser la propagation de l'information et donc la permission. Nous prévoyons de nous intéresser à la propagation du flux d'information afin de connaitre selon les

permissions accordées le but de l'usage de l'information. En d'autres termes, il faudrait savoir si la permission est utilisée pour assurer les fonctionnalités de l'application ou si elle est destinée à des usages inconnus des utilisateurs.

Bibliographie

[1] Gartner, (2015) *Gartner Says by 2017, Mobile Users Will Provide Personalized Data Streams to More Than 100 Apps and Services Every Day.* Disponible en ligne à : http://www.gartner.com/newsroom/id/2654115 (Consulté : 24 Octobre 2017).

[2] IDC, (2015) IDC: *Smartphone OS Market Share.* Disponible en ligne à : http://www.idc.com/promo/smartphone-market-share/os (Consulté : 25 Octobre 2017).

[3] Distimo, (2011) *The battle for the most content and the emerging tablet market*. Disponible en ligne à : http://www.distimo.com/blog/2011_04_the-battle-for-the-most-content-and-the-emerging-tablet-market. (Consulté : 24 Octobre 2017).

[4] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. *Pscout: analyzing the android permission specification*. In ACM Conference on Computer and Communications Security (CCS), pages 217–228.

[5] A. Porter Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. *Android Permissions: User Attention, Comprehension, and Behavior*. In Proceedings of the 8th Symposium on Usable Privacy and Security, 2012.

[6] A. P. Felt, K. Greenwood, and D. Wagner. *The Effectiveness of Application Permissions*. In Proceedings of the USENIX Conference on Web Application Development (WebApps), 2011.

[7] D. Barrera, H. Kayacik, P. van Oorschot, and A. Somayaji. *A methodology for empirical analysis of permission-based security models and its application to android*. In Proc. of the ACM conference on Computer and Communications Security (2010).

[8] D. Sufatrio, Tan, T. Chua, and V. Thing, (2015) *Securing Android: A Survey, Taxonomy, and Challenges*. *ACM Comput. Surv.* 47, 4, Article 58 (May 2015), 45 pages. DOI=http://dx.doi.org/10.1145/2733306

[9] Open Classrooms, *L'architecture d'Android*, Disponible en ligne à : https://openclassrooms.com/courses/creez-des-applications-pour-android/l-architecture-d-android (Consulté : 15 Novembre 2017).

[10] Wikipédia, *Noyau Linux*, Disponible en ligne à : https://fr.wikipedia.org/wiki/Noyau_Linux (Consulté : 20 Novembre 2017).

[11] Techotopia, *An overview of the Android Architecture*, Disponible en ligne à: http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture (Consulté: 22 Novembre 2017).

[12] Android developpers, *Building and Running*, Disponible en ligne à: https://stuff.mit.edu/afs/sipb/project/android/docs/tools/building/index.html (Consulté: 20 Novembre 2017).

[13] Android, (2017) *Application Security*, disponible en ligne à : https://source.android.com/security/overview/app-security (Consulté : 20 Novembre 2017).

[14] Android Intent. (2016) *Intents and Intent Filters | Android Developers.* Disponible en ligne à : http://developer.android.com/guide/components/intents-filters.html (Consulté : 24 Novembre 2017).

[15] Radoniaina ANDRIATSIMANDEFITRA RATSISAHANANA. *Caractérisation et détection de malware Android basées sur les flux d'information*. Thèse de doctorat.

[16] Android Developpers, *Requesting Permissions*, Disponible en ligne à : https://developer.android.com/guide/topics/permissions/requesting.html

[17] Android developpers, <*uses-permission*>, disponible en ligne à : https://developer.android.com/guide/topics/manifest/uses-permission-element.html (Consulté : 20 Novembre 2017).

[18] Android Permission, (2016) <*permission*> | *Android Developpers*. Disponible en ligne à : http://developer.android.com/guide/topics/manifest/permission-element.html (Consulté : 22 Novembre 2017).

[19] Android Marshmallow, (2016) *Android – Marshmallow*. Disponible en ligne à : https://www.android.com/versions/marshmallow-6-0/?gclid=CjwKEAiAgKu2BRDu1OGw3-KXokwSJAB_Yy2QVhdelA0Y2awKEncwMwxQp8_wTKfH8FlO_KG2N9pWWBoCp_Pw_wcB (Consulté : 23 Novembre 2017).

[20] R. Amadeo, (2015) *Android 6.0 Marshmallow, thoroughly reviewed.* Disponible en ligne à : http://arstechnica.com/gadgets/2015/10/android-6-0-marshmallow-thoroughly- (Consulté : 18 Novembre 2017).

[21] E. Chin, A. Felt, K. Greenwood, and D. Wagner. (2011) *Analyzing inter-application communication in Android.* Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11, pp. 239-252, ACM Publish.

[22] D. Octeau, P. McDaniel , S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. L. Traon., (2013) *Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis.* In Proceedings of the 22Nd USENIX Conference on Security, pp. 543–558, 2013.

[23] J. Rubin, M. I. Gordon, N. Nguyen, and M. Rinard. *Covert communication in mobile applications*. In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

[24] W. Enck, P. Gilbert, B. Chun, P. Cox, J. Jung, P. McDaniel, and A. N. Seth. (2010) *TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones.* In Proc. Of the 9th USENIX Symp. On Operating Systems Design and Implementation (OSDI) (2010).

[25] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "*Checking app behavior against app descriptions*," in Proceedings of the 36th International Conference on Software Engineering, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 1025–1035.

[26] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. WHYPER: *Towards automating risk assessment of mobile applications*. In USENIX Security Symposium, pages 527–542, 2013.

[27] M. Frank, B. Dong, A. P. Felt, and D. Song. *Mining permission request patterns from Android and Facebook applications*. In Proc. of the Data Mining (ICDM), 2012 IEEE 12th International Conference on, 2012.

[28] Y. Zhou and X. Jiang. *Dissecting Android malware: Characterization and evolution*. In IEEE Symposium on Security and Privacy (SP), pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.

[29] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "*Mining apps for abnormal usage of sensitive data*," in Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, vol. 1, May 2015, pp. 426–436.

[30] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. *FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps*. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, pages 259–269, New York.

[31] V. Moonsamy, J. Rong, S. Liu, G. Li, L. Batten, *Contrasting permission patterns between clean and malicious Android applications*, Future Generation Computer Systems Springer International Publishing, Cham, 2013, pp. 69–85.

[32] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro and Y. Zhang. *Investigating the relationship between price, rating, and popularity in the Blackberry World App Store*. Information and Software Technology 87 (2017) 119–139.

[33] Matthieu FOUQUIN (2004) *DATAMINING C4.5 – DBSCAN*. Disponible en ligne à : http://devezeb.free.fr/downloads/ecrits/datamining.pdf (Consulté : 20 Novembre 2017).

[34] Edutech Wiki, *Clustering et classification hiérarchique en texte mining*, http://edutechwiki.unige.ch/fr/Clustering_et_classification_hi%C3%A9rarchique_en_text_mining#Regroupement_Hi.C3.A9rarchique (Consulté : 20 Novembre 2017).

[35] A. Hinneburg and D. Keim, "*An efficient approach to clustering in large multimedia databases with noise*," in Proc. 4th Int. Conf. Knowledge Discovery and Data Mining (KDD'98), 1998, pp. 58–65.

[36] M. Ester, H. peter Kriegel, and X. Xu, "*A density-based algorithm for discovering clusters in large spatial databases with noise*," in International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[37] Androguard. Disponible en ligne à : http://code.google.com/p/androguard/ (Consulté : 20 Novembre 2017).

[38] S. Mohamed Aymen, B. Omar, A. Hani, and S. Houari, "*Mining multilevel API usage patterns*," in International Conf. on Software Analysis, Evolution, and Reengineering. IEEE, 2015.

[39] Apktool. Disponible en ligne à : https://ibotpeaches.github.io/Apktool/ (Consulté : 20 Novembre 2017).