Défauts d'intégrité contextuelle liés à la collecte de données personnelles par des applications de médias sociaux sur Android

par

Taïna BOUGANIM

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE
LA MAÎTRISE AVEC MÉMOIRE EN GÉNIE
DES TECHNOLOGIES DE L'INFORMATION
M. Sc. A.

MONTRÉAL, LE 30 JUILLET 2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Jean-Marc Robert, directeur de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Roger Champagne, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Chamseddine Talhi, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 25 JUILLET 2018

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à remercier toutes les personnes et organismes qui ont contribué à la réalisation de ce mémoire.

Tout d'abord, j'adresse mes remerciements à mon directeur de maîtrise, M. Jean-Marc Robert, directeur du département de génie logiciel et des TI et professeur à l'ÉTS, pour m'avoir offert l'opportunité de faire ce mémoire sous sa direction et pour sa disponibilité tout au long de ce projet.

J'exprime ma reconnaissance envers l'organisme Mobi-Centre et l'ÉTS pour leur soutien financier, ainsi que le bureau des Relations Internationales de l'INSA et de l'ÉTS pour m'avoir permis de réaliser une mobilité dans le cadre d'un double-diplôme.

Je remercie également les membres du jury de l'INSA et de l'ÉTS pour avoir accepté d'évaluer mon mémoire.

Je tiens à remercier mes parents pour le soutien inconditionnel dont ils ont fait preuve tout au long de mes études, et tout particulièrement mon père qui m'a donné le goût de la recherche, mes amis pour rendre l'expatriation douce et joyeuse et Nathan pour partager cette aventure outre-Atlantique avec moi.

Enfin, je remercie l'ensemble du corps enseignant de l'École de technologie supérieure et de L'INSA Centre-Val de Loire sans qui, il n'aurait pas été possible de mener à bien ma formation de maîtrise.

DÉFAUTS D'INTÉGRITÉ CONTEXTUELLE LIÉS À LA COLLECTE DE DONNÉES PERSONNELLES PAR DES APPLICATIONS DE MÉDIAS SOCIAUX SUR ANDROID

Taïna BOUGANIM

RÉSUMÉ

Les applications de médias sociaux démultiplient le volume de données contextuelles collectées corrélant numérique et environnement physique en temps réel. Cela a de nombreuses conséquences, encore peu appréhendées, pouvant porter atteinte à la vie privée de leurs usagers.

Nous explorons la notion d'« intégrité contextuelle » lors de la collecte de données de dix applications hégémoniques de médias sociaux sur la plateforme mobile Android. À savoir, nous évaluons l'écart entre les attentes d'un utilisateur et l'accès effectif aux ressources du mobile et à ses données personnelles.

Ce mémoire présente trois études complémentaires :

1. une étude a priori qui permet de cerner les pratiques de collecte annoncées (politiques de confidentialité, autorisations et permissions);
2. une analyse pratique qui instrumente le mobile pour recueillir la fréquence et les circonstances d'accès aux ressources de localisation et de messagerie texte, régulées par des autorisations;
3. une étude de la viabilité d'une solution permettant à l'utilisateur de paramétrer les permissions concédées aux applications en fonction du contexte d'utilisation.

Nous mettons en lumière des défauts d'intégrité contextuelle, tant au niveau de l'étude a priori (politiques peu claires, incohérences, défauts structurels), que de l'étude pratique (captage de la localisation à chaque seconde pour certaines applications). La solution proposée permet de mitiger ces problèmes et a peu d'impact sur les fonctionnalités des applications.

Mots-clés : Android, applications mobiles, système d'autorisations, politique de confidentialité, Intégrité contextuelle

CONTEXTUAL INTEGRITY BREACHES BY SOCIAL MEDIA APPLICATIONS WHILE COLLECTING PERSONAL DATA ON ANDROID PLATFORM

Taïna BOUGANIM

ABSTRACT

Social media applications leverage the amount of contextual data collected, correlating in real time digital and physical environment. This has many consequences, still little apprehended, that could affect the privacy of their users.

We explore the notion of "contextual integrity" when collecting data from ten hegemonic social media applications on the Android mobile platform. In other words, we measure the gap between users' expectations and effective access to their mobile resources and personal data.

This report presents three complementary studies:

1. a study of the announced collection practices (privacy policies, authorizations and permissions) to identify contextual integrity breaches;
2. a practical analysis for which we instrumented the mobile to collect the frequency and circumstances of access to location and text messaging resources, both regulated by authorizations;
3. and a proof of concept of a solution allowing the user to parameterize permissions such that they are granted to the applications according to the context of use.

We highlight contextual integrity breaches, both at the level of the announced collection practices study (unclear policies, inconsistencies, structural problems), and the practical study (capture of the location every second for some applications). The proposed solution mitigates these issues and has little impact on application functionality.

Keywords : Android, Mobile Applications, Authorization System, Privacy Policy, Contextual Integrity

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

3G	Troisième Génération
4G	Quatrième Génération
API	Application Programming Interface
APK	Android Package Kit
ART	Android RunTime
CSV	Comma Separated Values
DNS	Domain Name System
GID	Group Identifier
GPS	Géo-Positionnement par Satellite
FTC	Federal Trade Commision
ICC	Inter-Component Communication
iOS	iPhone Operating System
IPC	Inter-Process Communication
IP	Internet Protocol
JVM	Java Virtual Machine
MMS	Multimedia Messaging service
PID	Process Identifier
RGPD	Règlement Général sur la Protection des Données
SD	Secure Digital
SIP	Session Initiation Protocol
SE	Système d'Exploitation
SMS	Short Message Service

SQL	Structured Query Language
UID	User Identifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol
WAP	Wireless Application Protocol
Wi-Fi	Wireless Fidelity
ZIP	Zoning Improvement Plan

INTRODUCTION

Initialement épiphénomène d'Internet, les médias sociaux sont aujourd'hui incontournables. Utilisés par trois milliards de personnes à travers le monde, soit 40 % de la population mondiale, (We Are Social Ltd & Hootsuite Inc., 2017), les médias sociaux sont devenus de puissants outils d'influence, d'information et d'échanges en tout genre : politique, loisirs, jeux, mode, affaires, etc. Cela s'explique en partie par l'essor du Web mobile et la généralisation du téléphone intelligent et de ses applications : 93 % des utilisateurs (2,78 milliards de personnes) accèdent aux médias sociaux depuis leur mobile (We Are Social Ltd & Hootsuite Inc., 2017). Ces applications démultiplient le volume de données contextuelles collectées corrélant numérique et environnement physique en temps réel. Cette fusion des médias sociaux en ligne et du mobile dans le monde réel a de nombreuses conséquences, encore peu appréhendées par les usagers, pouvant porter atteinte à leur vie privée (Kayes & Iamnitchi, 2015; Krubhala, Niranjana, & Sindhu Priya, 2015).

Les avancées technologiques et la sophistication des téléphones intelligents, réels ordinateurs de poche dotés de fonctionnalités propres à leur caractère mobile (géolocalisation, boussole, accéléromètre, gyromètre, cartographie numérique, etc.) ont rendu possible une collecte massive, précise et ininterrompue des activités de leur propriétaire. Dans le contexte des applications de médias sociaux où l'exposition de soi est un élément intrinsèque du service, les données collectées par la version mobile de ces applications sont donc, par essence, potentiellement très sensibles (localisation, affiliation politique, activité, etc.). Cette métamorphose des possibilités de collecte, stockage et traitement des données par les services en ligne n'a cependant pas fondamentalement changé les pratiques en termes d'information et consentement de l'utilisateur (Kayes & Iamnitchi, 2015; Obar & Oeldorf-Hirsch, 2016; Shilton, 2009).

Sur Android, plateforme mobile la plus utilisée au monde, l'utilisateur n'a que deux indicateurs des pratiques de collecte d'information par les applications téléchargées sur le marché applicatif officiel Google Play : les autorisations, description textuelle des droits

d'accès aux ressources du mobile (mémoire, capteurs, connectivité, photos, etc.) requises par l'application et un hyperlien vers les politiques de confidentialité. De précédentes études ont proposé des mécanismes permettant d'évaluer le risque d'atteinte à la vie privée en extrayant ces méta-informations, à savoir les autorisations (Chia, Yamamoto, & Asokan, 2012; Frank, Dong, Felt, & Song, 2012; Qu et al., 2014) et les politiques de confidentialité (Coen, King, & Wong, 2016; Yu, Luo, Liu, & Zhang, 2016). Cependant, l'évaluation du risque nécessite un contexte – comment et quand les ressources sensibles sont utilisées – difficile à appréhender pour des utilisateurs néophytes (Shklovski, Mainwaring, Skúladóttir, & Borgthorsson, 2014) et même pour ceux qui sont aguerris (Spensky et al., 2016).

Il s'agit dès lors d'interroger la pertinence du modèle actuel de consentement utilisateur, ses limites et améliorations possibles au regard des problèmes soulevés par les applications de médias sociaux captant avec toujours plus d'acuité nos activités quotidiennes, nos intérêts, nos interactions, etc.

Pour cela, nous proposons d'examiner le modèle de consentement sur la plateforme Android, ayant plus de 70 % du marché mobile, appliqué à dix applications du top 15 des médias sociaux au Canada (We Are Social Ltd & Hootsuite Inc., 2017) figurant également dans le top 25 mondial : Facebook, YouTube, Facebook Messenger, Instagram, Snapchat, WhatsApp Messenger –appelée WhatsApp dans la suite du rapport–, Skype, Pinterest, Twitter et LinkedIn. Notons ici que dans la littérature, les études considèrent souvent un échantillon représentatif de l'écosystème mobile et donc s'attèlent à la tâche de détecter des applications malveillantes. Le contexte de ce mémoire est plus précis : les dix applications étudiées sont considérées de confiance, de notoriété publique et sont toutes des médias sociaux. Il s'agit ici de cerner leurs pratiques de collecte à partir des informations accessibles à l'utilisateur (politiques de confidentialité et autorisations) et ensuite d'étudier leur adéquation avec la pratique. Pour ce faire, nous réalisons une expérience exploratoire afin d'identifier des situations où la collecte défie les attentes utilisateur et/ou discordes avec les pratiques annoncées. Nous avons développé à cet effet des outils de traces qui nous permettent de surveiller les « comportements » de collecte des applications étudiées et d'y associer un

contexte d'exécution. Enfin, nous étudions la viabilité d'une solution permettant à l'utilisateur de paramétrer les permissions concédées à ses applications en fonction du contexte d'utilisation.

Le mémoire est structuré comme suit. Le chapitre 1 présente le système Android, son architecture et le modèle de sécurité adopté puis se concentre sur les mécanismes du système Android liés à la protection de la vie privée. Une fois ces notions préliminaires introduites, le chapitre 2 précise le problème et décrit la méthodologie adoptée dans cette étude. Dans le chapitre 3, nous présentons une revue de la littérature scientifique et technique en rapport avec la protection de la vie privée dans le contexte des téléphones mobiles. Les chapitres 4, 5 et 6 correspondent respectivement à l'analyse a priori des politiques de confidentialité et des autorisations, à l'étude pratique des accès aux ressources en instrumentant le mobile et à l'étude de la viabilité de la solution permettant de restreindre l'accès aux ressources en fonction du contexte d'utilisation. Ces trois chapitres constituent donc le cœur de ce travail. Enfin, nous concluons ce mémoire en le résumant et en apportant quelques considérations et suggestions de recherches à entreprendre pour corroborer cette étude.

CHAPITRE 1

NOTIONS PRÉLIMINAIRES

1.1 Le système Android

Les informations de cette section sont basées sur les recueils de Elenkov (2014) et de Murphy (2017) ainsi que sur la documentation Android en ligne (https://developer.android.com/index.html). La référence au système Android dans l'ensemble du mémoire, lorsqu'elle n'est pas précisée, est relative à la version Android Nougat (version 7.0, APK version 24).

1.1.1 Architecture

Android est un système d'exploitation (SE) mobile au code source libre (*open source*) basé sur le noyau Linux et développé essentiellement en Java. Il est constitué de cinq couches principales (*Voir* figure 1.1) :

- Le **noyau** fournit des pilotes pour le matériel, le réseau, l'accès au système de fichier et la gestion des processus;
- La couche native de l'espace utilisateur contient un processus **init** (qui lance tous les autres processus) un ensemble de **démons natifs** et de **bibliothèques logicielles natives** (écrites en C ou C++).
- L'**environnement d'exécution ART** (*Android Runtime*) implémentant une machine virtuelle Java (*JVM*) pour mobile et des bibliothèques d'exécution Java.
- La couche des intergiciels (*middleware*) comporte des **services système** (téléphonie, connectivité, affichage, etc.), et le **cadre d'application** (*framework)* Android. Ce cadriciel définit des **interfaces de programmation** (APIs) utilisées par les applications, s'exécutant dans la couche supérieure, pour interagir avec les ressources système (Ex. : l'API pour utiliser la caméra). La majorité des fonctionnalités au-dessus du noyau du SE Android sont implémentées en tant que services système, cependant, elles ne sont

accessibles par les applications que via des classes de façade appelées gestionnaires (*manager*).

- La couche applicative se compose **d'applications intégrées** (système et constructeur) et **d'applications installées** par l'utilisateur.

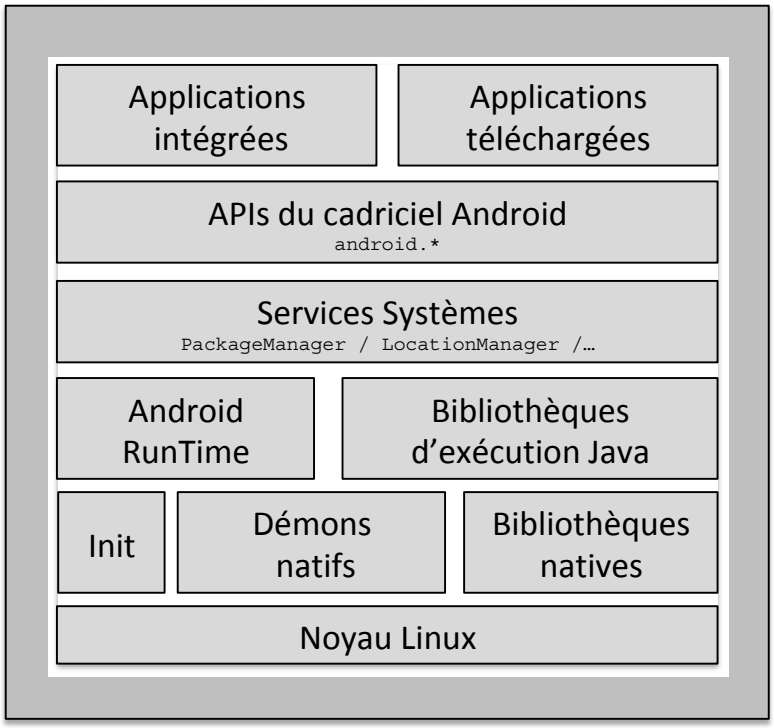

Figure 1.1 Architecture du Système d'Exploitation Android

Les applications Android sont bâties à partir de plusieurs composants qui fournissent des fonctionnalités spécifiques : **les activités** (*Activities*) définissent une interface utilisateur; **les services** (*Services*) effectuent un traitement en tâche de fond; **les fournisseurs de contenu** (*Content Providers*) implémentent le stockage et la récupération de données de bases de données reposant sur SQLite; et les **récepteurs de diffusion** (*Broadcast Receivers*) reçoivent des messages diffusés par le système ou par d'autres composants et réagissent de forme adéquate. Par exemple, lancer l'application au démarrage de l'appareil, lors de la réception du message *« ACTION_BOOT_COMPLETED ».* Chaque progiciel (*package*) d'une application Android inclut un fichier **AndroidManifest.xml** définissant les composants de

l'application afin que ceux-ci puissent être reconnus et initialisés par le système. Les composants ont comme balise racine < *application* > et sont respectivement définies dans les balises < *activity* >, < *service* >, < *provider* > et < *receiver* > (*Voir* figure 1.2).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.android.exampleApplication">
      <uses-sdk android:minSdkVersion="23" />
      <application
      android:label="Application_Exemple" >
            <activity
            android:name="Activité_Exemple" />
            <service
            android:name="Service_Exemple" />
            <provider
             android:name="Fournisseur_de_contenu_Exemple"
             android:authorities="com.exemple.android.exempleApplicationProvider"/>
            <receiver
             android:name="com.exemple.android.Recepteur_de_diffusion_Exemple">
                        <intent-filter android : name="Filtre_d_intent_Exemple">
                        <action android:name="android.intent.action.EXEMPLE" />
                        </intent-filter>
            </receiver>
      </application>
</manifest>
```

Figure 1.2 Exemple de fichier AndroidManifest.xml

Les applications peuvent communiquer entre elles via différents mécanismes : les IPC standards Linux (*Inter-Process Communication*), les interfaces de communication (*sockets*) Internet et les Binders. Les Binders sont des IPC légers permettant une communication entre les composants d'applications, dite **communication inter-composants** (ICC). Bâtis au-dessus des Binders, les **Intents** sont des messages émis par une application ou un service système et délivrés à l'aide du mécanisme ICC Android. Plus précisément, un Intent est un message asynchrone qui contient une tâche à déléguer à un composant tiers, et s'il y a lieu, l'URI (Uniform Resource Identifier) de données à traiter par le composant cible. Les Intents peuvent être envoyés explicitement aux composants nommés ou implicitement en utilisant une **action** prédéfinie. Pour la résolution des Intents implicites, les composants utilisent des **filtres d'Intents**, à savoir des marqueurs, pour signifier qu'ils sont aptes à traiter des actions prédéfinies (*Voir* figure 1.2). Android redirigera alors automatiquement les Intents implicites vers les composants appropriés (*Voir* figure 1.3). Si plusieurs applications (composants)

peuvent gérer un même Intent implicite, l'utilisateur est invité à sélectionner l'application à utiliser. Enfin, les applications et le système peuvent utiliser les **Intents en diffusion** afin de notifier qu'un évènement est survenu à l'ensemble des récepteurs de diffusion publics du mobile.

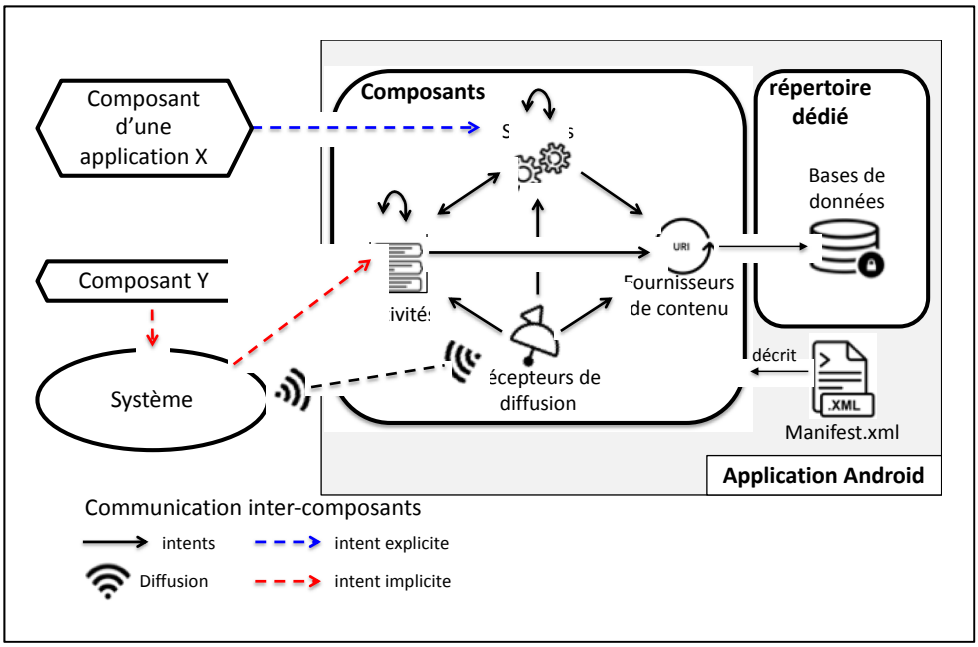

Figure 1.3 Communication inter-composants au sein d'une même application et entre applications

1.1.2 Modèle de sécurité

Android exige que chaque application soit signée numériquement par une clé privée associée à un certificat numérique, les certificats auto-signés étant acceptés. Cette **signature** est utilisée afin d'associer l'application à son développeur, et vérifier l'intégrité du progiciel de l'application lors de son installation. La signature est également utilisée lors de l'attribution de permissions par le système (*Voir* section 1.3.3).

Android fait usage des fonctionnalités de sécurité offertes par le noyau Linux comme moyen pour identifier les applications et isoler leurs ressources. Il est important de rappeler que

Linux est un système d'exploitation multi-utilisateurs et multitâches : il a donc des mécanismes d'isolation des processus et des utilisateurs, afin d'éviter des interférences indésirables. Les processus y ont des espaces d'adressage distincts, et un processus ne peut pas accéder directement à la mémoire d'un autre processus. Aussi un utilisateur n'a pas accès aux fichiers (exécutables ou non) d'un autre utilisateur, sauf autorisation explicite. Enfin, chaque processus s'exécute avec l'identité de l'utilisateur (UID : *User Id*) qui l'a démarré. Cet identifiant est donné soit à un utilisateur physique du système exécutant des commandes via le Shell, soit à un service système (démon) qui s'exécute en arrière-plan.

Sur Android, système d'exploitation conçu initialement pour des appareils personnels, l'UID est utilisé pour distinguer les applications : à l'installation, chaque application Android se voit assigner un UID unique et définitif. Aussi, chaque application s'exécute dans un processus dédié et se voit attribuer un répertoire privé auquel elle seule a les droits de lecture/écriture. Ainsi, les applications sont isolées à la fois au niveau du processus et au niveau des fichiers. Ce mécanisme d'isolement des applications est couramment appelé **« bac à sable »** *(sandboxing)*.

La **communication inter-composants** est régulée par le système via un mécanisme de **contrôle d'accès**. Ce mécanisme limite l'accès à des APIs protégées (ressources et fonctionnalités) du système ou à des composants d'applications tierces. L'objectif est de déterminer « qui » a « quel type d'accès » à « quelles ressources ». Ici, on s'intéresse au contrôle d'accès applicatif, à savoir pour une application donnée, comment sont régis les accès aux ressources du système ou d'applications tierces ?

Le mécanisme est basé sur des **permissions** : le système définit un ensemble de permissions nécessaires pour accéder à différentes ressources. Ainsi, une application voulant accéder à la ressource appareil photo devra impérativement avoir la permission *« android.permission. CAMERA ».* Notons ici qu'il est possible, pour une application, de créer des ressources (activité, service, récepteur de diffusion, fournisseur de contenu) et de les rendre disponibles à d'autres applications. Il suffit pour cela d'indiquer, dans le manifeste, l'attribut

exported = *true* ou d'instancier des filtres d'Intents avec la balise < *intent-filter* > dans la déclaration du composant concerné. On dit alors que le composant est public. Dans le cas des récepteurs de diffusion, dire que le composant est public signifie que ce dernier peut recevoir des Intents émis par des composants externes à son application. Le mécanisme de protection est alors le même, l'application définit des permissions, ou utilise celles définies par le système, afin de contrôler l'accès à ses ressources publiques.

1.2 Modèle Android de protection de la vie privée

Nous présentons dans cette section les étapes clés du modèle de protection de la vie privée sur Android (*Voir* figure 1.4). Nous détaillerons dans un premier temps son cadre légal. Puis nous étudierons le versant utilisateur du processus à savoir le système d'autorisations et de consentement. Enfin, nous expliquerons le versant technique à savoir la déclaration et validation des permissions.

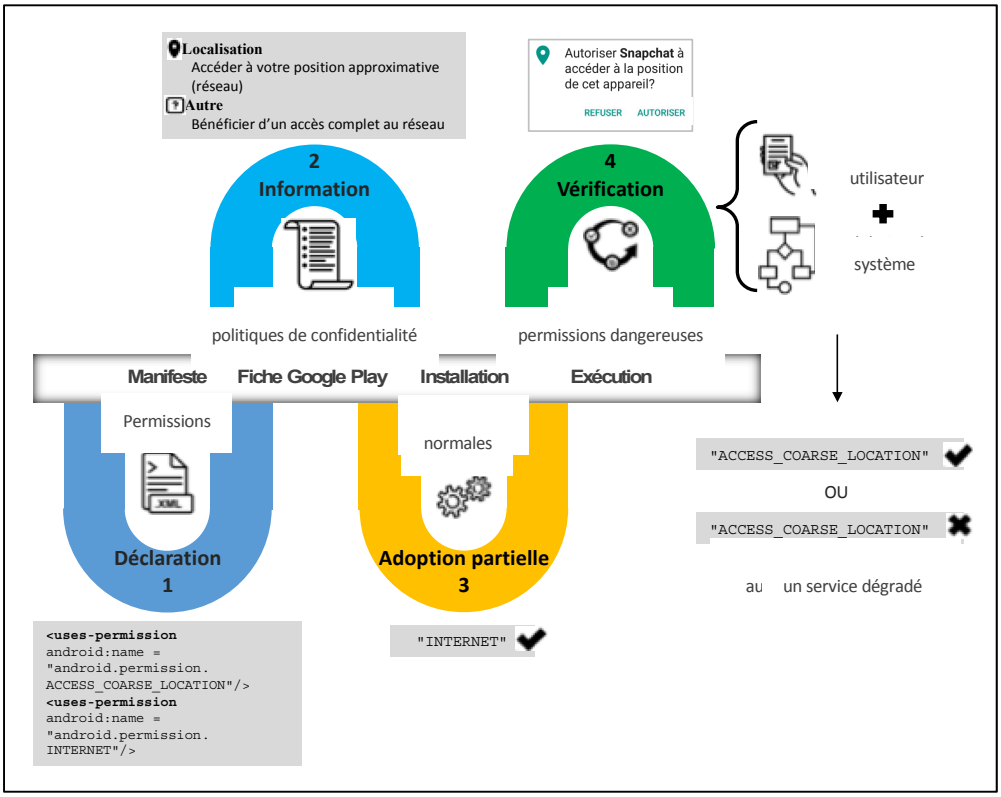

Figure 1.4 Processus Android de protection de la vie privée

1.2.1 Aspects juridiques : Politiques de confidentialité

Quand il est question de données personnelles et service en ligne, l'approche « **notification et consentement** » est la norme. Cette approche informe l'utilisateur des pratiques de confidentialité du service, à savoir de collecte et traitement des données personnelles, et donne à l'utilisateur le choix de s'engager, ou non, avec le service. Les médias sociaux mobiles ne dérogent pas à la règle. La demande de consentement se fait à l'inscription, nécessitant une action de la part de l'utilisateur : cocher la case « *J'ai lu et accepte les conditions d'utilisation du service et politiques de confidentialité* », appuyer sur un bouton « *En cliquant sur créer un compte, vous acceptez nos conditions et indiquez que vous avez lu notre politique de confidentialité* » ou autre. D'autre part, Google Play, marché applicatif officiel d'Android, exige que les applications, traitant des informations personnelles ou sensibles, fassent preuve de **transparence** et publient des **règles de confidentialité** sur la fiche Google Play Store de l'application, ainsi que dans l'application en elle-même (Google Play, 2017). Ces règles doivent décrire dans le détail les méthodes de collecte, d'utilisation et de partage des données de l'utilisateur et l'application doit limiter la pratique (collecte, utilisation partage de données) au cadre. Enfin, la loi applicable peut imposer d'autres mesures et pénalités en termes d'atteinte à la vie privée des utilisateurs.

Dans ce mémoire, nous nous concentrons uniquement sur la collecte des informations personnelles ou sensibles, et, plus précisément sur les informations recueillies sur l'utilisateur. Les informations qu'il saisit sont en dehors du cadre de l'étude[1].

[1] Déterminer si une information est saisie ou captée est assez complexe et se prête à une analyse plus approfondie dans la suite du mémoire (*Voir* sections 2.1.2 et 4.2.3).

1.2.2 Notifications et consentement utilisateur

1.2.2.1 Le système d'autorisations

Comme principal mécanisme de protection de la vie privée des utilisateurs, Android utilise un **modèle sophistiqué d'autorisations**. Sur les plateformes mobiles, le système d'autorisations régit la manière dont les applications accèdent à certaines ressources, telles que les informations à caractère personnel, les données du mobile et du réseau ou encore les données des capteurs (caméra, GPS, etc.). Les applications Android s'exécutent dans un « bac à sable », zone isolée qui n'a pas accès au reste des ressources du système, sauf si des autorisations d'accès leur sont accordées. Le processus d'autorisation est en deux étapes. Dans un premier temps, le développeur déclare que son application requiert des **permissions** d'accès à des ressources du mobile (mémoire, capteurs, connectivité, etc.) dans le manifeste de l'application. Dans un second temps, l'utilisateur est invité à examiner et accorder ou refuser l'accès aux ressources lors de l'installation ou de l'exécution de l'application.

En pratique, une **autorisation** est une courte proposition générique telle que « Lire le journal d'appel » ou « Accéder à votre position précise (GPS et réseau) » de demande d'accès à une ressource, ici le journal d'appel ou les coordonnées GPS du mobile, sans information supplémentaire de son utilité pour le fonctionnement de l'application. Plus précisément, une autorisation Android est une traduction en langage naturel d'une permission d'accès à une ressource destinée à informer l'utilisateur des besoins de l'application.

1.2.2.2 Changements dans le système d'autorisations Android

Les versions antérieures à Marshmallow, la sixième version Android sortie au cours de l'été 2015, avaient un système d'autorisations binaire qui présentait à l'installation la liste des autorisations requises par l'application. Si l'utilisateur refusait de concéder une ou plusieurs des autorisations listées, sa seule option était de renoncer à installer l'application (Android Developpers : System Permissions, 2017). De précédentes études (Felt, Chin, Hanna, Song, & Wagner, 2011) ont révélé que ce système était **inefficace** en termes de **responsabilisation**

de l'utilisateur final en relation à la vie privée : l'utilisateur n'était pas en mesure de comprendre ni de contrôler, l'accès des applications à ses données privées. Vient s'ajouter à cela la prolifération d'applications demandant plus d'autorisations que nécessaire (Gorla, Tavecchia, Gross, & Zeller, 2014) et le problème de **l'accoutumance** de l'utilisateur. Comme l'utilisateur ne comprend pas toujours pourquoi une application nécessite une donnée ou une fonctionnalité spécifique, il n'est pas en mesure d'identifier les cas de sur-réclamation d'autorisations. Aussi, les demandes d'autorisations systématiques à l'installation altèrent fortement la capacité d'attention rationnelle et de compréhension des individus. Cette altération met à mal la capacité de donner l'aval (consentement) et de contrôler la légitimité de la demande. Dès lors, une majorité d'utilisateurs acceptent les autorisations sans les lire afin d'avoir accès aux services ou fonctionnalités des applications (Felt, Ha, et al., 2012).

Le nouveau modèle d'autorisations Android (version Marshmallow et ultérieures) introduit un changement majeur en demandant les autorisations lors de l'exécution de l'application. La documentation officielle d'Android pour les développeurs (Android Developpers : Permissions Overview, 2018) classe les autorisations en deux catégories en termes de sécurité et risque d'atteinte à la vie privée : normale et dangereuse. Une **autorisation** est considérée **dangereuse** quand elle permet l'accès à des informations sensibles : données stockées sur l'appareil ou données captées à caractère personnel (Localisation, SMS, Contacts, etc.). Une autorisation est dite **normale** lorsqu'elle accède à des ressources pas ou peu sensibles, avec un risque d'atteinte à la vie privée de l'utilisateur ou au fonctionnement des autres applications du mobile très limité. Ainsi, les autorisations normales sont **accordées d'office** par le système et ne sont pas révocables. Elles doivent être cependant notifiées dans le manifeste, sans quoi elles ne sont pas accordées. Cela permet notamment de respecter l'exigence de transparence de Google Play et Android et d'informer un utilisateur soucieux de sa vie privée. Les permissions dangereuses sont **demandées** la première fois que l'application les requiert **à l'exécution**. Cela permet une **contextualisation** de l'autorisation et ainsi une meilleure compréhension de l'utilisateur. De plus, l'utilisateur peut, à tout moment, **consulter**, accorder ou **révoquer** les autorisations dangereuses d'une application

dans Paramètres/Applications/Autorisations. Ainsi l'utilisateur est responsabilisé et peut contrôler les applications susceptibles d'affecter sa vie privée. Le modèle est basé sur le principe de minimisation des privilèges (Andriotis, Sasse, & Stringhini, 2016) : il suppose que les applications pourront fonctionner avec **service dégradé** pour les fonctionnalités nécessitant des permissions récusées par l'utilisateur que nous appellerons dans la suite un « **niveau basique** ». Cela se traduit, par exemple, par une localisation approximative ou l'absence de synchronisation des contacts de l'application avec le carnet d'adresses du mobile.

1.2.2.3 Analyse du nouveau système d'autorisations

Le nouveau modèle d'autorisations, des versions Marshmallow et ultérieures, a néanmoins quelques zones d'ombres. Premièrement le « niveau basique » des applications, permettant aux utilisateurs de limiter les autorisations d'accès sans renoncer à l'application, n'est pas implémenté pour une grande majorité des applications (Andriotis et al., 2016). L'utilisateur est ainsi notifié que l'application a été conçue pour des versions antérieures et qu'elle risque de ne plus fonctionner s'il lui révoque des autorisations. En second lieu, les demandes dynamiques d'autorisations ne sont affichées qu'une fois, la première fois que l'application a besoin de l'autorisation d'accès à la ressource sensible. L'effort de contextualisation est amoindri, en effet, il est peu probable que l'utilisateur ait déjà appréhendé le « comportement » de l'application et sache juger la pertinence et le bien-fondé de la demande. Aussi, même si la première fois l'accès à la ressource lui semble justifié, il n'est pas évident que ce soit le cas pour des demandes ultérieures (Wijesekera et al., 2015). Finalement, les **autorisations** sont **accordées par groupe**. À titre d'exemple, le groupe *Contacts* contient trois autorisations distinctes : « Lire vos contacts », « Modifier vos contacts » et « Rechercher des comptes sur l'appareil ». Si l'utilisateur octroie une autorisation du groupe X à une application, chaque fois que cette même application demande une autre autorisation du groupe X, le système la lui accordera immédiatement sans consulter l'utilisateur (Andriotis et al., 2016). Bien que cela soit une façon de remédier au problème de l'accoutumance en réduisant le nombre de décisions de l'utilisateur, il n'est pas évident que

ce dernier soit sollicité sur des autorisations concernant des données qui le préoccupent. La granularité des autorisations est contestable : quel est l'intérêt de présenter différentes autorisations spécifiques si l'aval doit être groupé ? L'objectif est probablement d'éviter la surprise et la confusion des utilisateurs, face à l'ampleur de la collecte de données sensibles sur leur mobile, observées par de précédentes études, et à terme une meilleure compréhension de cette pratique (Egelman, Felt, & Wagner, 2013; Felt, Ha, et al., 2012; Lin et al., 2012; Shklovski et al., 2014). Notons ici que la pertinence de classification d'une permission en dangereuse ou normale est, elle aussi, discutable, mais en dehors du cadre de cette étude.

1.2.3 Définition et vérification des permissions

Cette section présente le versant technique (développeur et système) du modèle de protection de la vie privée sur Android. Ses informations sont basées sur les recueils de Elenkov (2014) et de Murphy (2017) ainsi que sur la documentation Android en ligne (https://developer.android.com/index.html).

1.2.3.1 Définition des permissions

Les permissions d'accès définies par le système ont une nomenclature standardisée : *« android.permission.EXEMPLE »*. Aussi toute application peut déclarer dans son fichier AndroidManifest.xml les permissions que des applications tierces sont tenues d'avoir en vue d'interagir avec ses composants (activité, service, fournisseur de contenu ou récepteur de diffusion). Ces permissions ont généralement une nomenclature de type *reverse-DNS*[1] *« com.appExemple.EXEMPLE »*. Dans le manifeste, une permission est définie en utilisant la balise < *permission* >, et inscrite dans l'attribut *android : permission* du composant devant être protégé (*Voir* figure 1.5). Le cas des fournisseurs de contenu est un peu particulier : ils jouissent d'un contrôle d'accès plus flexible. Étant de réelles petites bases de données ces

[1] *reverse-DNS* : est une convention de dénomination en informatique basée sur des noms de domaine (DNS) enregistrés dont l'ordre des chaînes de caractères est inversé. Cette notation inversée est un moyen de réduire les collisions d'espace de nom, car tout nom de domaine enregistré est unique.

composants disposent de restrictions à granularité variable. Il leur est possible de n'accorder un accès à la base qu'en lecture via l'attribut *android : readPermission* ou en écriture via *android : writePermission*, ou plus précis encore, de n'accorder l'accès qu'à une entrée identifiée par une URI.

```
<permission
      android:name="com.appExemple.EXEMPLE"
      android:protectionLevel="signature" />
<application
      android:label="Application_A" >
      <service
            android:name= "Service_A"
            android:permission="com.appExemple.EXEMPLE"/>
                  …
</application>
```

Figure 1.5 Déclaration d'une permission

Chaque permission sur Android, qu'elle soit définie par le système, par une librairie ou une application tierce, est associée à un **niveau de protection**, via l'attribut *android : protectionLevel* de l'élément < *permission* > (*Voir* figure 1.5). Ce niveau définit comment la permission en elle-même sera accordée à une application la réclamant. Il existe quatre niveaux possibles :

- *normal* : niveau par défaut, la permission est non révocable et accordée de façon automatique par le système;
- *dangereux* : la permission est révocable et accordée par l'utilisateur à l'exécution. Une telle permission peut être ajoutée ou retirée dynamiquement de la liste des permissions accordées à une application;
- *signature* : la permission est accordée automatiquement par le système aux applications ayant été signées par la même clé privée (et donc appartenant au même développeur) que l'application ayant déclaré cette permission;
- *signature* | *privileged* : la permission est non révocable et accordée automatiquement par le système aux applications faisant partie de l'image système ou signées avec la même clé qu'une application intégrée ayant déclaré la permission. Pour rappel une application intégrée est une application système ou constructeur.

1.2.3.2 Vérification des permissions

Le contrôle de permissions est appliqué à plusieurs niveaux sur Android. Les composants de niveau supérieur (applications et services système) interrogent le gestionnaire de progiciels pour déterminer les autorisations qui sont présentement attribuées à l'application appelante et décident d'accorder ou non l'accès. Les composants de niveau inférieur (démons natifs) n'ont pas accès au gestionnaire de progiciels et utilisent les identifiants affectés à un processus, à savoir l'identifiant du processus (PID, UID, GIDs *Voir* ci-après) afin de déterminer les privilèges d'accès à accorder au processus. L'accès aux ressources système est régulé par le noyau et se base également sur les identifiants affectés à un processus. Plus en détail, chaque application Android se voit assigner, à son installation, un UID unique et s'exécute dans un processus dédié. Lors de l'exécution de l'application, le gestionnaire de progiciels attribue au processus dédié un PID (*process Id*entifier) et l'UID de l'application. Si l'application a acquis des permissions, ces dernières sont associées à des GIDs (*Group Identifiers*) et affectées au processus. Cette association permet donc aux composants bas niveau de réaliser une vérification dynamique des permissions de l'application appelante au niveau de leur processus dédié, sans en avoir une connaissance effective.

L'accès effectif à une ressource est accordé si et seulement si l'application appelante possède les permissions requises par la méthode de l'API ou le composant appelé. Ces permissions sont spécifiées dans le manifeste de l'application appelante en utilisant la balise < *uses-permission* > (*Voir* figure 1.6).

```
<uses-permission android:name="com.appExemple.EXEMPLE" />
<uses-permission android:name= "android.permission. BLUETOOTH" />
…
<application
      android:label="Application_B" >
            …
</application>
```

Figure 1.6 Demande de permissions d'accès

1.2.3.3 Permissions dangereuses

Dans le cas où il s'agit d'une **permission dangereuse**, à savoir, pouvant être validée ou refusée par l'utilisateur, la présence dans le manifeste n'est qu'une condition nécessaire, mais pas suffisante pour le contrôle d'accès. Lors du premier accès effectif à la ressource, l'application appelante est tenue de demander le **consentement explicite de l'utilisateur**. L'application présente alors une boîte de dialogue par groupe d'autorisations requises (*Voir* figure 1.7), et si l'utilisateur refuse, la même boîte de dialogue sera présentée à chaque fois que l'application tente d'accéder à la ressource. Afin d'éviter l'agacement de ce dernier, dès la deuxième requête d'un groupe d'autorisations donné il est ajouté une option de refus définitif (*Voir* figure 1.8).

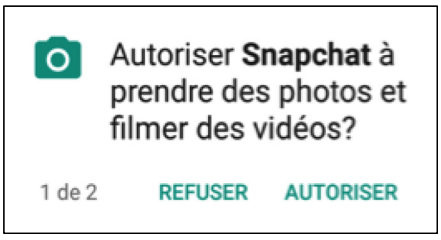

Figure 1.7 Première demande d'autorisations

Figure 1.8 Demandes d'autorisations ultérieures

En pratique quand l'utilisateur donne son consentement, la ou les permissions sous-jacentes sont ajoutées à la liste des permissions acquises de l'application par le gestionnaire de progiciels. Si à l'inverse, l'utilisateur va dans Paramètres/Applications/Autorisations et révoque un groupe d'autorisations précédemment accordé, la ou les permissions sous-jacentes sont retirées de cette liste. Il y a une bijection entre permissions et autorisations dangereuses, néanmoins, comme les autorisations sont accordées par groupe, lorsqu'une application demande par exemple les permissions *« android.permission.RECEIVE_SMS »* et *« android.permission.SEND_SMS »* dans son manifeste, l'aval ou le refus sera donné pour l'ensemble du groupe *Messagerie texte*. Ainsi pour une décision utilisateur, plusieurs permissions peuvent être impactées. Il existe neuf groupes de permissions / autorisations

dangereuses contenant chacun entre une et sept permissions / autorisations pour la version 7.0 d'Android (*Voir* tableau 1.1). Notons ici que l'association réalisée entre permissions et GID (pour attribuer des privilèges d'accès aux processus) est en fait l'association d'une permission à son groupe.

Tableau 1-1 Permissions dangereuses et leur groupe sur Android 7.0

Groupe de Permissions	**Permissions**	**Autorisations**
AGENDA	READ_CALENDAR, WRITE_CALENDAR	- Consulter les évènements d'agenda ainsi que les informations confidentielles - Ajouter ou modifier des évènements d'agenda et envoyer des e-mails aux invités à l'insu du propriétaire
APPAREIL PHOTO	CAMÉRA	- Prendre des photos et filmer des vidéos
CONTACTS	READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS	- Lire vos contacts - Modifier vos contacts - Rechercher des comptes sur l'appareil
LOCALISATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION	- Accéder à votre position précise (GPS et réseau) - Accéder à votre position approximative (réseau)
MICROPHONE	RECORD_AUDIO	- Enregistrer des fichiers audio
TÉLÉPHONE	READ_PHONE_STATE, CALL_PHONE READ_CALL_LOG, WRITE_CALL_LOG, ADD_VOICEMAIL, USE_SIP, PROCESS_OUTGOING_CALLS	- Voir l'état et l'identité du téléphone - Appeler directement des numéros de téléphone - Lire le journal d'appels - Modifier le journal d'appels - Ajouter des messages vocaux - Faire et recevoir des appels SIP - Transférer les appels sortants
CAPTEURS CORPORELS	BODY_SENSORS	- Accéder aux capteurs corporels (comme le moniteur de fréquence cardiaque)
MESSAGERIE TEXTE	SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS	- Envoyer et afficher des messages texte - Recevoir des messages texte (SMS) - Voir les messages texte ou multimédias - Recevoir des messages texte (WAP) - Recevoir des messages texte (MMS)
STOCKAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE	- Lire le contenu de la carte SD - Modifier ou supprimer le contenu de la carte SD

Lorsqu'un utilisateur révoque un groupe de permissions à une application en cours d'exécution via le gestionnaire système de permissions, son processus dédié est arrêté. Cependant, l'application qui perd la permission n'en est pas informée. C'est pourquoi le guide pour développeur Android recommande d'utiliser la méthode *Context.checkSelf Permission(permission)* avant d'effectuer une opération sensible afin de s'assurer de bien avoir la permission requise.

1.2.4 Résumé

Le modèle de protection de la vie privée sur Android est une démarche de transparence déclarative (politiques de confidentialité et listes des autorisations) couplée à un système de contrôle d'accès impliquant **trois acteurs** :

- le **développeur** qui déclare que son application requiert des permissions d'accès à des ressources dans le manifeste;
- le **système** qui accorde, à l'installation, l'ensemble des permissions normales requises et contrôle, à l'exécution, si l'application détient la permission nécessaire (*Voir* figure 1.9);
- l'**utilisateur** qui est sollicité pour concéder, lors du premier accès de l'application à une ressource protégée, le ou les groupes d'autorisations dangereuses requises.

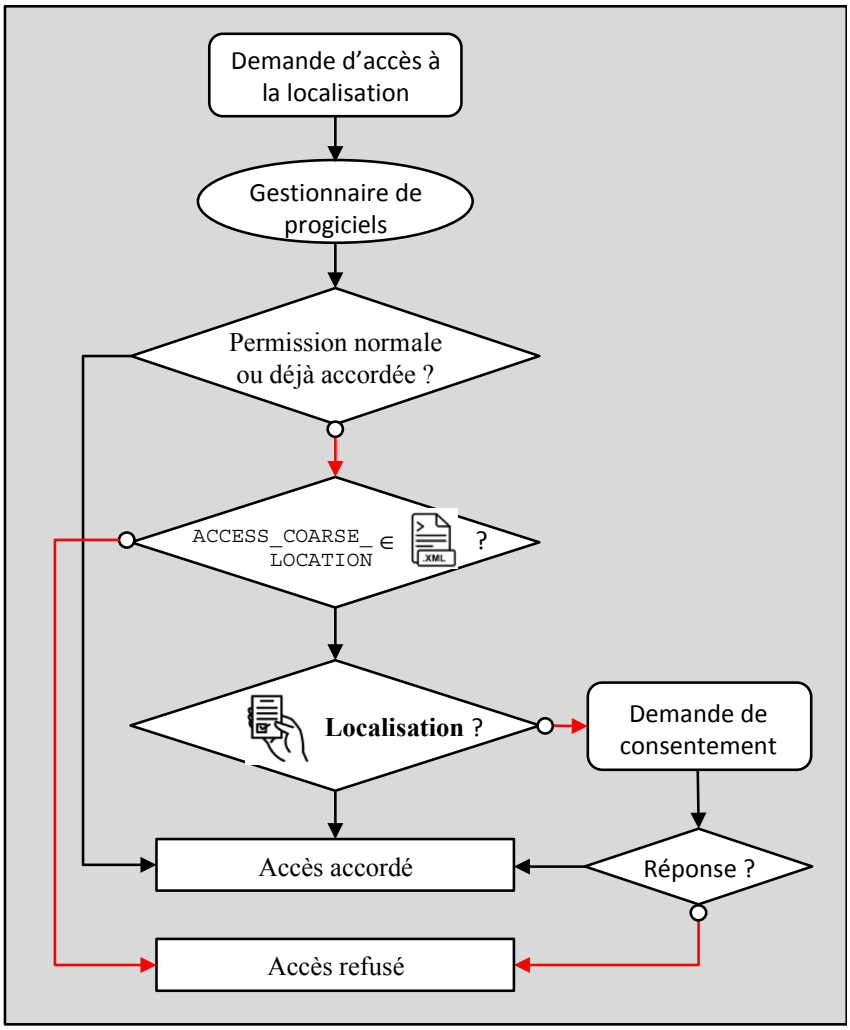

Figure 1.9 Vérification dynamique des permissions concédées à une application, exemple de *« Access_Coarse_Location »*

CHAPITRE 2

PROBLÈME ET MÉTHODOLOGIE

2.1 Problème étudié

La présente étude interroge la pertinence du modèle actuel de consentement utilisateur concernant la collecte d'informations dans le contexte des applications de médias sociaux sur mobiles Android. Sur la base de la théorie développée par Nissenbaum (2004), nous utilisons dans la suite de ce mémoire les termes « **défaut d'intégrité contextuelle** » pour décrire une collecte de données non conforme au modèle conceptuel des utilisateurs, à savoir un accès effectif à des ressources du mobile et aux données personnelles de l'utilisateur qui défie les attentes des utilisateurs. L'idée est donc de mettre en lumière les limitations des moyens d'information et de notification à l'utilisateur concernant la collecte de ses données personnelles (politiques de confidentialité, autorisations, permissions) et le défaut d'intégrité contextuelle qui en résulte (accès effectifs). Si tel est le cas quels sont les vecteurs d'actions possibles pour les utilisateurs et ces derniers sont-ils adaptés ?

Notons que, bien que l'étude se concentre sur les médias sociaux à des fins de comparaison, la démarche pourrait être étendue à d'autres catégories d'applications (jeux, sport, cartes et navigations, etc.).

2.1.1 Accès effectif aux ressources par les applications

Cette étude vise, entre autres, à déterminer la fréquence à laquelle une application donnée accède effectivement à des ressources protégées par des permissions. Pour cela, il est nécessaire de capturer chaque appel aux méthodes d'accès aux ressources du mobile ou données utilisateur.

Il est important de souligner qu'il n'existe pas d'associations entre méthode accédant à des ressources et permissions. Il est précisé dans la documentation Android (et Google) quand

une méthode publique d'une classe donnée requiert une permission. Cependant, il n'y a pas, pour une permission donnée, l'ensemble des méthodes accédant à la ressource qu'elle « protège ». Aussi, Android étant de source libre, il existe de nombreuses librairies tierces accédant à des ressources sensibles : répertorier tous les moyens d'accéder à une ressource se révèle donc un travail titanesque. De plus, une application peut utiliser des mécanismes, d'Intents et des permissions URI afin d'accéder à des ressources par l'intermédiaire d'une application tierce. Dès lors, comment comptabiliser les accès pour une application donnée à une ressource donnée ? Nous abordons ce problème plus en détail dans la méthodologie.

2.1.2 Collecte de données et médias sociaux

Dans cette étude lorsqu'il est question de collecte de données, les données saisies par l'utilisateur ne sont pas pertinentes : un utilisateur qui saisit des données en est conscient, il ne peut donc y avoir de défaut d'intégrité contextuelle pour ce genre de données. Cependant, les données captées sont moins perceptibles par l'utilisateur et donc sujettes à l'évaluation de l'écart entre attente et pratique. Dans le cadre des médias sociaux, cette différenciation entre données saisies / partagées et données captées / générées est parfois imperceptible. Il s'agit ici donc de définir un cadre plus précis : qu'entend-on par média social et pourquoi la distinction entre données saisies et données captées est ambigüe ?

Média social est un terme générique, utilisé le plus souvent au pluriel, regroupant toute plateforme basée sur les technologies du Web 2.0, qui vise à faciliter la création et le partage de contenu généré par les utilisateurs, la collaboration et l'interaction sociale (Grand Dictionnaire de Terminologie, 2018). Les applications mobiles de ces plateformes constituent un secteur applicatif particulièrement sensible, et ce pour deux raisons principales. Premièrement, les téléphones intelligents ont des fonctionnalités variées gérant de nombreuses données utilisateur (photographies, message texte, contact, etc.) et générant, pour chacune, un nombre accru de **métadonnées** très précises (date, heure, géolocalisation, expéditeur, compte associé, etc.). Deuxièmement, sur les médias sociaux l'exposition de soi et le partage sont des éléments intrinsèques du service. Dès lors, les applications de médias

sociaux proposent des fonctionnalités aux utilisateurs, souvent par défaut, afin de faciliter le partage d'informations captées : une photographie sciemment partagée peut être automatiquement géolocalisée; le carnet d'adresses du mobile peut être synchronisé avec le réseau de contacts de l'application; l'application peut être utilisée comme application de messagerie par défaut; etc. Ici, le verbe pouvoir suggère que l'utilisateur a le choix : il peut, via les paramètres de l'application ou les autorisations Android, concéder ou non à l'application les accès à ces données captées, à savoir respectivement ici localisations, contacts et messages standards (SMS/MMS). Ce choix est-il pour autant informé et réfléchi ? L'utilisateur est-il en mesure de comprendre et contrôler cette collecte de données captées ?

Pour y répondre, ce mémoire se concentre sur l'étude de dix applications hégémoniques de médias sociaux : Facebook, YouTube, Messenger, Instagram, Snapchat, WhatsApp, Skype, Pinterest, Twitter et LinkedIn. Il s'agit ici, dans un premier temps, de cerner les pratiques annoncées de collecte de données captées, à partir des informations accessibles à l'utilisateur (politiques de confidentialité et autorisations) et de souligner leur éventuelle inaptitude à informer convenablement l'utilisateur et donc avoir un consentement tangible. Par la suite, il est question d'établir un constat pratique, dans un contexte d'utilisation réaliste, de l'accès effectif aux ressources du mobile par ces dix applications, afin d'identifier de potentiels défauts d'intégrité contextuelle. Par souci de précision, et au vu du problème de comptabilisation des accès effectifs soulevé dans la section précédente, nous avons restreint l'étude à deux groupes d'autorisations emblématiques : *Localisation* et *Messagerie texte*. Enfin, si une application porte atteinte à la vie privée de l'utilisateur –dans le sens où elle accède à des données de manière en dérogeant aux attentes de l'utilisateur– nous étudions les vecteurs de réactions possibles pour ce dernier et leurs limitations.

2.2 Méthodologie

2.2.1 Identification d'une problématique

Le leitmotiv de la recherche dans le domaine de la protection de la vie privée sur le mobile est la prévention et la détection de fuite d'informations personnelles, à savoir l'identification

de comportement douteux et des risques en matière d'atteinte à la vie privée. De nombreuses solutions très techniques présentées en première section de la revue de littérature permettent d'atténuer ces risques automatiquement. Cependant, il nous semble ici pertinent de souligner que « porter atteinte à la vie privée » est une notion très subjective se basant sur la perception de l'utilisateur d'un comportement inapproprié des applications. Une solution viable doit donc inclure l'utilisateur dans le processus de protection de ses données personnelles. La deuxième section de la revue de littérature examine alors les diverses méthodes de sensibilisation des utilisateurs aux risques d'atteinte à leur vie privée. Ces recherches nous ont permis de porter un intérêt tout particulier aux limitations du système actuel de notification et consentement utilisateur via les politiques de confidentialité et les autorisations.

Le changement du système de permissions, introduit avec Android version 6, témoigne d'un abandon progressif d'un système trop permissif, au profit de la sécurité et de la protection des données. Cependant, il faillit à sa tâche car il ne tient pas compte de l'intégrité contextuelle, à savoir s'assurer que les accès effectifs à des ressources du mobile et données personnelles sont en accord avec les attentes de l'utilisateur. Cette étude cherche à mettre en lumière ce défaut d'intégrité contextuelle et tente de s'inscrire dans une démarche d'amélioration continue.

Afin de dégager une problématique réalisable dans le cadre d'un mémoire, nous avons décidé de restreindre l'étude à la collecte d'informations captées par des applications de confiance. Cela permet ainsi de retirer le biais lié à un comportement déviant dans la perception d'atteinte à la vie privée et de se concentrer uniquement sur la genèse des fuites d'information : leur génération / collecte. Remarquons ici que considérer une application « de confiance » relève également de la subjectivité de l'utilisateur. Nous avons donc fait l'hypothèse, de par leur grande adoption, utilisation et longévité que les applications de médias sociaux prépondérantes ont la confiance de la grande majorité de leurs utilisateurs. De plus, comme déjà mentionné, se concentrer sur un domaine applicatif permet une certaine cohérence et continuité d'analyse.

La méthodologie de recherche adoptée dans cette étude s'articule par la suite en trois phases : une étude a priori, une étude pratique et une démonstration de faisabilité.

2.2.2 Étude a priori

Premièrement, nous effectuons une analyse a priori permettant de souligner les obstacles à la compréhension de l'utilisateur et à un consentement averti. Pour cela, nous étudions les sections des politiques de confidentialité de chaque application se référant à la collecte de données captées. Nous examinons également les groupes d'autorisations requis par les applications et leurs permissions sous-jacentes. Nous portons un intérêt particulier aux autorisations dangereuses sur lesquelles l'utilisateur a, à ce jour, un moyen d'action limité : concéder ou révoquer le groupe d'autorisations sans distinction de cas d'usage. Les autorisations normales seront également étudiées pour leur rôle informatif (elles sont attribuées d'office par le système et sont non révocables).

2.2.3 Étude pratique

Par la suite, nous voulons évaluer l'adéquation entre collecte annoncée et accès effectifs. Le problème est qu'il n'existe pas, à notre connaissance, d'outils permettant d'enregistrer et quantifier les accès effectifs aux ressources protégées. Aussi, pour accéder à de telles informations il est nécessaire de posséder les droits administrateur sur le mobile. Ces droits sont généralement obtenus en débridant le mobile (*rooting*).

Dans une optique de faisabilité, nous restreignons ici l'étude pratique à deux groupes de permissions, *Localisation* et *Messagerie texte*, classées dangereuses par Android et considérées sensibles par les utilisateurs. Pour ce faire, nous avons développé un outil, ***PrivacyStats***, qui modifie le système d'exécution d'Android afin de capturer les accès aux données de localisation ou messagerie texte et les journalise, en se basant sur le code d'un projet de source libre *Xprivacy* (Bokhorst, 2012), module du cadriciel *Xposed* (Rovo89 & Tungstwenty, 2011).

Comme déjà expliqué en section 2.1.1, identifier les chemins d'accès à une ressource protégée par une application est une tâche difficile. Une première étape consiste donc à établir l'association entre les méthodes d'accès aux ressources ou données sensibles (appel API, Intent, etc.) et les permissions des groupes *Localisation* et *Messagerie texte*. Une fois ces méthodes identifiées, nous développons un code spécifique, dit un « *hook* », qui s'exécutera à l'appel et/ou au retour de chaque méthode afin de journaliser l'accès aux ressources de façon adéquate. Nous avons testé la validité de notre outil de capture / traces avec des ***applications*** faisant appel aux méthodes instrumentées. Pour cette étape, nous avons étudié les chapitres appropriés du livre *The Busy Coder's Guide to Android Development 8.9* (Murphy, 2017) qui passe en revue l'ensemble des techniques actuelles de développement d'applications Android. Le livre est particulièrement exhaustif, avec pas moins de 4436 pages d'explications et 395 exemples d'applications, dont le code source est accessible sur un répertoire *GitHub* (Murphy, 2017) mis à jour mensuellement. Il nous a semblé pertinent de prendre ce guide comme base de référence définissant l'ensemble des méthodes accédant aux ressources protégées couramment utilisées. Aussi, afin de développer des applications de test valides, nous avons adapté quelques-unes des applications fournies pour solliciter uniquement les méthodes d'intérêt. Nous avons sélectionné et adapté 32 applications qui nous permettaient de tester toutes les méthodes en lien avec des permissions dangereuses, avant de finalement nous restreindre à huit pour tester les méthodes associées aux groupes *Localisation* et *Messagerie texte*. Cet effort, bien que fastidieux, ne garantit malheureusement pas l'exhaustivité de notre solution, mais permet d'avoir une certaine confiance dans les résultats.

2.2.4 Démonstration de faisabilité

Enfin, nous cherchons un vecteur possible d'amélioration du système d'autorisations actuel qui permettrait à l'utilisateur de paramétrer les permissions concédées à ses applications en fonction du contexte d'utilisation. Nous étudions ainsi comment surveiller l'état d'une application (en premier plan, en arrière-plan, arrêtée) et s'il est possible de corréler les permissions d'accès acquises à cet état. En d'autres termes, nous cherchons à évaluer la

faisabilité d'un blocage des permissions d'une application en arrière-plan. Nous avons développé, à cet effet, une application ***BlockPermBG*** et lui avons attribué les droits systèmes. Par la suite, nous évaluons l'impact de cette suspension de droits d'accès sur la collecte des données captées et sur les fonctionnalités des applications bloquées.

2.2.5 Expérience

Nous avons réalisé notre expérience sur les versions des applications étudiées lors de l'analyse préliminaire. Nous avons donc empêché les mises à jour pour ne pas biaiser les résultats. Nous avons créé un persona féminin de 21 ans et constitué des réseaux de contacts sur les dix médias sociaux étudiés afin d'être dans des conditions d'expérimentation vraisemblables.

Après quelques tests préliminaires en situation, à savoir avec les dix applications de médias sociaux installées et actives sur le mobile, nous avons réalisé qu'il est extrêmement difficile d'analyser des traces sans un environnement contrôlé. Nous avons alors créé une application, ***ContextInfo****,* qui capture à une fréquence paramétrable le contexte d'exécution du mobile (Ex. : activités utilisateur en cours, applications installées, actives, en arrière-plan). L'idée est de réduire au maximum les biais liés aux erreurs de manipulation et de capter automatiquement les conditions expérimentales. Aussi, les fichiers de traces issus de tests sont filtrés par nos ***scripts Python*** et ***requêtes SQL*** afin d'extraire les données utiles pour notre analyse, automatisant ainsi la récupération et le traitement des traces générées dans une seule et même base de données.

L'expérience suit une approche holistique et déductive, nous faisons une analyse statistique incrémentale de la fréquence d'appel, pour chaque application, aux différentes méthodes instrumentées et tentons d'identifier les catalyseurs de ces appels (activité utilisateur, sollicitation de fonctionnalités spécifiques, ouverture de l'application, etc.). De fait si l'utilisateur utilise la fonctionnalité « partager localisation en direct », l'appel à la méthode *requestLocationUpdates* semble justifié et appréhensible par l'utilisateur, nous cherchons

donc à identifier des situations où ce lien de causalité fait défaut. Les traces sont recueillies régulièrement jusqu'à obtention de résultats satisfaisants. Nous avons tout d'abord testé le mobile sans les applications étudiées puis, chacune séparément, et enfin toutes ensembles. Cela permet de mieux comprendre la corrélation entre appels et contexte, et réduire les erreurs d'interprétation des résultats. Aussi, lors des tests, si nous observons qu'une application a un « comportement » régulier et déterministe (c.-à-d. nous avons déterminé et compris son comportement) nous la désinstallons du mobile afin de nous concentrer sur les applications présentant potentiellement plus de risques de défaut d'intégrité contextuelle. De même, si certaines configurations, avec par exemple le téléphone en veille, génèrent des traces sans intérêt nous les retirons des protocoles de tests subséquents.

La méthodologie de recherche adoptée dans notre étude pratique est résumée dans la figure 2.1, ci-dessous.

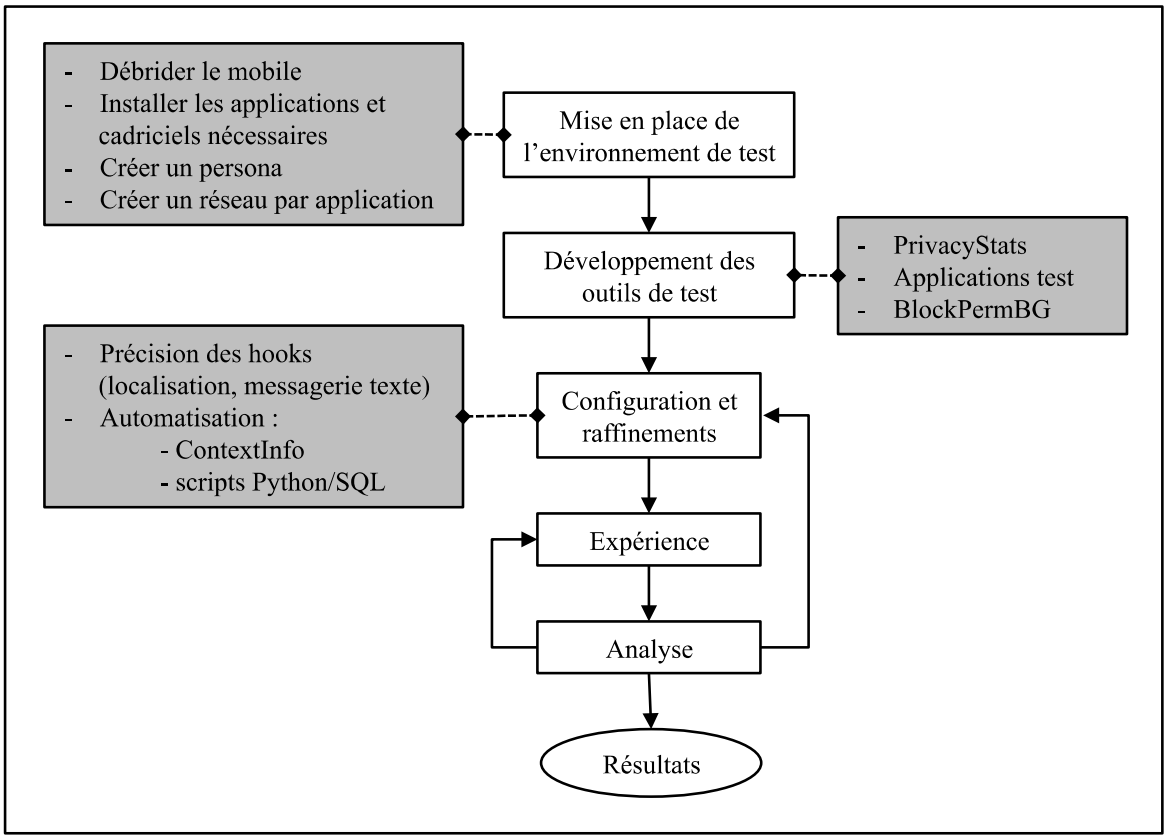

Figure 2.1 Méthodologie de l'étude pratique

CHAPITRE 3

REVUE DE LITTÉRATURE

3.1 Détection et prévention de fuite d'information sur Android

De nombreux efforts ont été mis en œuvre ces dernières années pour détecter et prévenir la fuite d'information à caractère personnel des téléphones intelligents. Les méthodes sont diverses (Haris, Haddadi, & Hui, 2014) et peuvent être classées grossièrement en cinq catégories : l'analyse statique, l'analyse de flux dynamique, l'analyse de flux hybride, l'analyse de flux réseau et la modification du modèle d'exécution des applications.

3.1.1 Analyse de flux statique

L'analyse de flux statique permet une détection a priori de fuite d'information d'une application donnée par analyse de son code source ou son pseudocode (*bytecode)*, code binaire compilé composé d'instructions exécutable par la machine virtuelle Dalvik (ancêtre de l'Android RunTime). Il existe différentes méthodes telles que la modélisation du cycle de vie de l'application (Flowdroid : Arzt et al., 2014), ou l'analyse sémantique du flux des données entre les sources et les points de fuite identifiés (DroidSafe : Gordon et al., 2015). Ces méthodes ont néanmoins quelques inconvénients, elles nécessitent le code source de l'application, ne prennent pas en compte les conditions d'exécution (interactions utilisateur, code chargé dynamiquement) et leur déploiement est complexe. De fait, l'analyse statique doit être faite, pour chaque application, en amont de la distribution sur les marchés applicatifs.

3.1.2 Analyse de flux dynamique

L'analyse de flux dynamique permet une détection et notification en temps réel (à l'exécution) des fuites d'information de données provenant de sources identifiées comme sensibles (Taintdroid : Enck et al. 2014; Uranine : Rastogi, Qu, McClurg & Chen, 2015;

TaintEraser : Zhu, Jung, Song, Kohno & Wetherall, 2011). Cependant, ces solutions sont très coûteuses en termes de ressources et souffrent d'important taux de faux positifs et faux négatifs. Aussi, Rastogi et al. (2015) proposent de reconditionner les applications ce qui signifie abdiquer à leurs mises à jour automatiques et aux mécanismes de partage des ressources ainsi qu'une perte de garantie constructeur.

3.1.3 Analyse de flux hybride

L'analyse de flux hybride combine l'analyse statique pour identifier les fonctions qui sont potentiellement dangereuses et une analyse dynamique pour filtrer et confirmer les conditions qui déclenchent de réelles fuites d'information (Scandroid : Fuchs, Chaudhuri, & Foster, 2009; Smartdroid : Zheng et al., 2012). Ces solutions combinent les avantages et défauts des solutions d'analyse de flux statique et dynamique.

3.1.4 Limitations des solutions d'analyse de flux d'information

La précision et l'efficacité de toutes les solutions d'analyse de flux d'information (statique, dynamique et hybride) reposent sur les listes de sources et de puits (points de fuite : *sink*) des données sensibles. En général, elles ne contiennent que quelques méthodes d'API Android connues. Dès lors, des sources et des puits moins connus ne sont pas détectés par ces outils d'analyse (Rasthofer, Arzt, & Bodden, 2014). De plus, elles sont confrontées aux problèmes de code impénétrable (remplacement d'identifiants et de labels mnémoniques par des labels non signifiants) (Continella et al., 2017).

3.1.5 Analyse du flux réseau

L'analyse de flux réseau identifie la fuite d'information ou le comportement douteux d'application en interceptant le trafic réseau via VPN puis en l'analysant (Recon : Ren, Rao, Lindorfer, Legout, & Choffnes, 2016; Privacyguard : Song & Hengartner, 2015). Les inconvénients majeurs de ce type de solutions sont l'impossibilité de garantir la complétude d'identification de fuites et la non-résistance à l'obfuscation et au chiffrement du trafic

(Continella et al., 2017). De fait, l'analyse n'est pas viable si les applications utilisent des techniques d'obfuscation, qui rendent le code impénétrable tout en le gardant pleinement fonctionnel, ou si le trafic est chiffré.

3.1.6 Modification du modèle d'exécution

La modification du modèle d'exécution englobe différentes catégories de solutions dont les plus notables sont :

- le remplacement de données sensibles par des données factices lorsqu'elles sont requises par des applications (Beresford, Rice, Skehin, & Sohan, 2011; Hornyack, Han, Jung, Schechter, & Wetherall, 2011);
- de permettre à l'utilisateur de bloquer ou restreindre les données accédées par les applications (PMP : Agarwal & Hall, 2013; FlaskDroid : Bugiel, Heuser, & Sadeghi, 2013; Riskmon : Jing, Ahn, Zhao, & Hu, 2014);
- un blocage de bibliothèque tierce (Chitkara, Gothoskar, Harish, Hong, & Agarwal, 2017).

Ces solutions nécessitent que l'utilisateur fasse un compromis entre protection de sa vie privée et utilisabilité des applications dont les fonctionnalités peuvent être affectées par les modifications du modèle d'exécution. De plus, de par leur complexité, ces solutions ne sont accessibles qu'a des utilisateurs ayant un important bagage technique (Agarwal & Hall, 2013; Chitkara et al., 2017). Aussi, beaucoup d'entre elles restent à l'état de prototype couplé à une étude faisabilité (Beresford et al., 2011; Bugiel et al., 2013; Hornyack et al., 2011; Jing et al., 2014) et ne peuvent pas être déployées en l'état.

3.2 Sensibilisation des utilisateurs aux risques d'atteinte à leur vie privée

Les méthodes des cinq catégories citées ci-dessus sont pour la plupart aussi complexes à appréhender que le problème auquel elles tentent de répondre : identifier et éviter l'atteinte à la vie privée des usagers de téléphones intelligents par des applications malveillantes. Ce décalage entre recherche et pratique a incité des chercheurs à apporter une attention

particulière à la sensibilisation de l'utilisateur aux risques et à l'aide à apporter pour prendre des décisions avisées en termes de vie privée. Ce domaine d'étude, en plein essor, a différents angles d'approches.

3.2.1 Production participative

Certaines études se basent sur la **production participative** (*crowdsourcing)*, une pratique qui consiste à faire réaliser une tâche par un grand nombre de personnes en faisant appel à l'intelligence collective pour créer du contenu, développer une idée, résoudre un problème ou réaliser un projet innovant (GDT, 2018). Les chercheurs utilisent ainsi les différents paramétrages des autorisations concédées aux applications des contributeurs afin de comprendre les attentes des utilisateurs en termes de respect de la vie privée et d'en créer un modèle conceptuel (Lin et al., 2012); de fournir des recommandations de protection de la vie privée spécifiques aux applications étudiées (Agarwal & Hall, 2013; Chitkara et al., 2017); ou encore, d'étudier l'impact sur l'utilisabilité de ces applications et de trouver le meilleur compromis (Ismail, Ahmed, Kapadia, & Reiter, 2015). La production participative est également utilisée pour obtenir des jeux de données annotées de politiques de confidentialité non structurées à des fins d'apprentissage machine supervisé, d'évaluation, d'analyse ou de sensibilisation des utilisateurs finaux (Oltramari et al., 2017; Sadeh et al., 2013; Wilson et al., 2016). Des chercheurs ont également utilisé la production participative à des fins de validation de leur outil générant automatiquement une description fidèle et lisible de la collecte d'informations par les applications Android, à partir d'une analyse statique du code source, permettant d'assister les développeurs dans la rédaction des politiques de confidentialité (Yu, Zhang, Luo, & Xue, 2015).

3.2.2 Apprentissage machine

De nombreuses études se basent sur des méthodes d'**apprentissage machine** pour permettre des résultats à grande échelle. Elles identifient un problème lié à la protection de la vie privée sur mobile et utilisent des outils et méthodes d'apprentissage machine pour y remédier. Par exemple, Wang, Hong, et Guo (2015) abordent le problème de compréhension médiocre du

système d'autorisations par les utilisateurs. Les chercheurs proposent donc un outil d'inférence automatisée afin de déterminer l'objectif d'utilisation des autorisations Android. Ren et al. (2016) ainsi que Rasthofer et al. (2014) traitent du problème de la sur réclamation d'autorisations et des méthodes détournées permettant de passer outre les mécanismes de sécurité mis en place. Plus précisément, Rasthofer et al. (2014) présentent le fonctionnement et les multiples validations de Susi, un outil de détection et catégorisation automatisée des méthodes responsables des fuites d'information sur Android. Ren et al. (2016) présentent ReCon, une solution multi-plate-forme actuellement déployée sur le nuage informatique (*Cloud*), utilisant l'apprentissage machine appliquée au monitorage de flux réseau pour prédire et notifier les utilisateurs de fuites d'informations à caractère personnel. Une interface utilisateur permet de visualiser et potentiellement bloquer ces fuites.

Jing et al. (2014) et Bilogrevic et al. (2016) s'attaquent au problème de la charge décisionnelle liée à la protection de la vie privée dans les systèmes mobiles. La profusion de facteurs et d'informations contextuelles à considérer altère la capacité d'attention rationnelle des individus. Ces études expliquent que les utilisateurs ne sont pas en mesure de faire des choix avisés dans le cadre du système d'autorisations Android (Jing et al., 2014) et de partage d'informations sur les réseaux sociaux mobiles (Bilogrevic et al., 2016). Pour pallier cela, Jing et al. (2014) ont développé Riskmon, un cadriciel qui assiste l'utilisateur dans l'évaluation du risque d'atteinte à la vie privée et le paramétrage des autorisations concédées aux applications Android en se basant sur le contexte et les préférences de l'utilisateur. Bilogrevic et al. (2016) proposent quant à eux SPISM (*Smart Privacy-aware Information Sharing Mechanism*), un système semi-automatisé de partage d'information sur les réseaux sociaux mobiles permettant de réduire la charge décisionnelle et de trouver l'équilibre souhaité par l'utilisateur entre vie privée, utilité et commodité.

3.2.3 Traitement automatique du langage naturel

Une branche en plein essor de l'apprentissage machine et de l'intelligence artificielle, le traitement automatique du langage naturel, est également mis à contribution par les

chercheurs afin de sensibiliser l'utilisateur aux risques et l'aider à prendre des décisions avisées en termes de vie privée. Le projet Usable Privacy Policy (Sadeh et al., 2013) se base sur les avancées du traitement automatique du langage naturel, et la production participative pour extraire des informations pertinentes de façon semi-automatique des politiques de confidentialité existantes. L'idée de ce projet, à l'inverse des initiatives existantes (P3P, Do Not Track, etc.) du domaine, est un déploiement de solutions indépendantes des opérateurs de sites Web et d'applications mobiles. Il a pour but de concevoir des interfaces pour présenter le contenu des politiques de confidentialité dans des formats plus compréhensibles par les utilisateurs finaux, ainsi que de développer un cadriciel de contrôle par les utilisateurs de la collecte et de l'utilisation de leurs données personnelles (Oltramari et al., 2017; Zimmeck et al., 2017).

3.2.4 Limitations des gestionnaires d'autorisations Android

De nombreuses études utilisateurs ont permis d'identifier, dans le début des années 2010, un écart important entre l'attente des utilisateurs et le comportement réel des applications mobiles en termes d'accès aux ressources du mobile et collecte des données personnelles (Balebako, Jung, Lu, Cranor, & Nguyen, 2013; Felt, Ha, et al., 2012; Thurm & Kane, 2010). Les utilisateurs ignoraient bien souvent l'étendue de ces pratiques, et lorsqu'ils en étaient informés beaucoup exprimaient des réserves et inquiétudes à ce propos (Felt, Egelman, & Wagner, 2012; Lin et al., 2012). Comme vu précédemment, le système d'autorisations binaire des versions antérieures à Android 6 était inefficace pour responsabiliser les utilisateurs finaux en matière de vie privée. Une majorité d'entre eux acceptent les autorisations sans les lire afin d'avoir accès aux services ou fonctionnalités des applications (Felt, Ha, et al., 2012). Wijesekera et al. (2015) ont effectué une étude sur le terrain avec 36 utilisateurs Android pour explorer la notion d'intégrité contextuelle : détecter les situations où des applications accèdent à des ressources protégées alors que les utilisateurs ne s'y attendent pas. Ils ont, pour cela, journalisé le nombre de vérifications de permissions, supposé en rapport avec le nombre d'accès effectif des applications des participants pendant une semaine puis réaliser des entrevues individuelles afin d'identifier les situations dans

lesquelles les utilisateurs auraient voulu bloquer l'accès. Ils identifient les mêmes problèmes que les précédentes études : compréhension médiocre de la pertinence d'une demande d'autorisation, accoutumance et manque d'attention rationnelle résultant à un aval non légitime et donc à un défaut d'intégrité contextuelle.

Le nouveau système tente de remédier à ces problèmes en introduisant des demandes d'autorisations par groupe logique à l'exécution. Cela permet une contextualisation de la demande ainsi qu'une réduction du nombre de demandes (Andriotis et al., 2016). Aussi Android encourage les développeurs à spécifier un objectif dans les boîtes de dialogue de demande d'autorisations. Cependant, cela peut se révéler contreproductif, car des explications vagues peuvent rendre l'utilisateur suspicieux et le dissuader d'accorder l'autorisation (Shih, Liccardi, & Weitzner, 2015).

Enfin de récentes études suggèrent que le gestionnaire d'autorisations a encore d'importantes lacunes : c'est un outil de contrôle limité, car il ne permet que les révocations d'autorisations jugées dangereuses par Android ou les développeurs, et non par l'utilisateur. Il est alors probable que les problèmes d'accoutumance et défaut d'intégrité contextuelle persistent. De plus, le système ne prend pas en considération la visibilité de l'application demandeuse (Wijesekera et al., 2015) : une demande d'accès par une application en arrière-plan est souvent mal comprise par l'utilisateur et est donc une atteinte à l'intégrité contextuelle. L'effort de contextualisation n'est pas suffisant : l'utilisateur est convié à accorder l'autorisation lors du premier accès, mais il peut être surpris de constater que l'application continue d'avoir accès à la ressource dans des contextes différents. Enfin, le gestionnaire d'autorisations manque d'informations cruciales pour aider les utilisateurs à prendre des décisions de confidentialité éclairées (Almuhimedi et al., 2015; Liu et al., 2016) telles que : pourquoi une permission donnée est nécessaire, à quelle fréquence l'application accède à une ressource spécifique, etc. De fait, informer les utilisateurs du but probable de la demande d'autorisation d'une application, basée sur de l'analyse de son code, permet d'éclairer la prise de décision en matière de vie privée (Lin, Liu, Sadeh, & Hong, 2014; Shih et al., 2015; Tan et al., 2014; Wang et al., 2015).

3.2.5 Aide et stimulus à la conscientisation de l'utilisateur

Des chercheurs ont exploré des solutions permettant d'alerter les utilisateurs et de les aider à gérer leur vie privée pour les applications déjà installées sur leurs appareils (Almuhimedi et al., 2015; Chitkara et al., 2017; Liu et al., 2016; Ren et al., 2016).

Une première technique consiste en présenter des *nudges*, dits « coup de pouce » en français, à savoir des messages d'alerte incitant l'utilisateur à réagir. Almuhimedi et al. (2015) ont conçu des *nudges* qui informent les utilisateurs du nombre d'accès effectifs à des informations personnelles (la localisation par exemple) et les invites à revoir leur paramétrage d'autorisations. Ils ont constaté que les *nudges* aident fortement à la compréhension des comportements des applications et l'appréhension des risques d'atteinte à la vie privée et donc à un ajustement responsable et réfléchi des autorisations concédées aux applications.

Une deuxième approche consiste à proposer à l'utilisateur un gestionnaire d'autorisations amélioré. C'est par exemple le cas de la solution déjà citée Riskmon (Jing et al., 2014), le cadriciel qui assiste l'utilisateur dans l'évaluation du risque d'atteinte à la vie privée. Pour évaluer le risque, RiskMon se base sur trois classes d'information : les méta-informations des applications installées (popularité et réputation), la cohérence entre les fonctionnalités de base de l'application évaluée et les communications inter-composants qu'elle requiert, et les préférences de sécurité saisies de l'utilisateur. Il automatise ainsi le procédé de révocation des autorisations à risque, en s'assurant que cela impacte le moins possible les fonctionnalités des applications concernées, et dans le cas où le comportement est très suspect il invite l'utilisateur à modifier les autorisations lui-même ou désinstaller l'application. Liu et al. (2016) proposent une solution qui utilise l'apprentissage machine afin d'établir des profils de préférences utilisateur en termes de protection de la vie privée. Le modèle est instruit et validé par production participative basée sur des révisions utilisateurs des autorisations. Dans cette solution, les *nudges* font également partie du processus d'amélioration continue du modèle : ils permettent d'informer l'utilisateur sur le nombre d'accès aux données privées

dans les derniers jours et l'objectif inféré de l'accès. Ainsi, ils stimulent une révision du paramétrage des autorisations et garantissent l'adéquation entre les paramètres et les réelles préférences utilisateur. Ainsi ce gestionnaire d'autorisation amélioré permet de diminuer la charge décisionnelle de l'utilisateur, grâce à l'apprentissage machine et la production participative, tout en le sollicitant et l'impliquant dans la gestion des autorisations, via les *nudges*.

Une troisième approche consiste à proposer des gestionnaires de confidentialité basés sur un contexte distinct du système d'autorisations. C'est le cas de la solution déjà citée, ReCon, utilisant l'apprentissage machine appliqué à la surveillance de flux réseau permettant à l'utilisateur de visualiser et potentiellement bloquer les fuites de données à caractère personnelles (Ren et al., 2016). On peut également citer ProtectMyPrivacy (Chitkara et al., 2017), une solution qui détecte les accès critiques aux données sensibles à l'exécution et détermine si l'accès est effectué par une bibliothèque tierce ou par l'application en elle-même. Les chercheurs constatent qu'un ensemble de 30 bibliothèques sont responsables de plus de la moitié des accès aux données privées. L'idée est alors de proposer un gestionnaire qui bloque ce type d'accès aux bibliothèques et non aux applications.

3.2.6 Conclusion

Un grand nombre de travaux dans le domaine de la sécurité mobile et de la recherche sur la vie privée ont porté sur le développement d'outils d'analyse automatisée. Cependant, distinguer si l'accès à certaines ressources sensibles est nécessaire, comprise et acceptée par l'utilisateur ne peut se faire qu'avec l'implication de ce dernier dans le processus de contrôle des risques d'atteinte à sa vie privée. Ce mémoire s'inscrit donc dans une recherche axée sur la sensibilisation et responsabilisation de l'utilisateur. Nous tentons de mettre en lumière les limitations et améliorations possibles du cadriciel Android pour aider l'utilisateur dans les décisions relatives à la collecte de ses données personnelles.

CHAPITRE 4

ANALYSE A PRIORI

4.1 Introduction

Cette analyse porte sur les indicateurs des pratiques de collecte d'informations disponibles sur la plateforme Android et son marché applicatif officiel Google Play, à savoir, les politiques de confidentialité, les autorisations et les permissions du manifeste. Elle vise à identifier les obstacles à la compréhension de l'utilisateur concernant la collecte de données et les défauts de légitimité du consentement utilisateur subséquent.

4.2 Politiques de confidentialité

4.2.1 Définitions et généralités

Une politique de confidentialité doit informer les utilisateurs sur la collecte, le traitement, le stockage et éventuellement la transmission de leurs données par le site Web, l'application mobile ou par tout autre service avec lequel il s'engage. Cela inclut également d'informer les utilisateurs de tout choix qu'ils pourraient avoir concernant ces pratiques. En d'autres termes, une politique de confidentialité est un contrat qui nécessite le consentement de la source des données : l'utilisateur.

Le cadre légal de la norme « notification et consentement » suppose que les utilisateurs sont des acteurs rationnels capables de lire les documents juridiques pour évaluer les pratiques de confidentialité de sites Web et d'applications mobiles. Pour se conformer à la loi sans inquiéter l'utilisateur, les sites Web / applications mobiles traitant des données personnelles utilisent des artifices permettant d'obtenir son consentement. Les politiques de confidentialité sont « présentées » lorsqu'un individu se connecte au service pour la première fois et bien souvent, la présentation se résume à un hyperlien menant au document. Cependant, l'utilisation du site Web, ou le téléchargement de l'application subséquent constituent un

consentement de facto aux politiques du service. Aussi certains prestataires de services *« se réservent le droit de modifier »* ladite politique sans notifier au préalable l'utilisateur qui devra donc *« consulter régulièrement »* le document et *« en continuant à utiliser le service »* il signifiera son *« acceptation de ces modifications»* (Instagram Inc.).

De multiples études ont confirmé l'évidence : ces politiques sont longues (McDonald & Cranor, 2008), vagues (Fei Liu, Fella, & Liao, 2016), souvent écrites dans un langage qui dépasse l'entendement d'un utilisateur lambda (Frederick Liu, Wilson, Schaub, & Sadeh, 2016; Reidenberg et al., 2015) et donc ignorées (Cate, 2006; Obar & Oeldorf-Hirsch, 2017). Dès lors, pourquoi les considérer comme étant partie intégrante du système de notification et de consentement utilisateur ? D'un point de vue légal, c'est le document qui fixe le cadre des pratiques touchant aux données des clients d'une entreprise ou utilisateurs d'un service, et qui fait foi en cas de litige. D'autre part, les avancées dans le domaine de l'intelligence artificielle, de l'apprentissage machine et du traitement automatique du langage naturel permettent d'envisager, dans un futur proche des outils permettant l'extraction de données utiles et la visualisation de politiques de confidentialité de forme plus accessible et compréhensible par les utilisateurs finaux, sans mettre à contribution les développeurs d'applications ou opérateurs du Web (Sadeh et al., 2013).

Il n'est pas aisé d'écrire des politiques de confidentialité adéquates, de ce fait il est courant que les auteurs d'une politique de confidentialité ne soient pas les développeurs de l'application mobile (ou du site Web). De plus, de nombreuses applications utilisent des bibliothèques tierces, leur politique de confidentialité est donc tenue de couvrir également les pratiques de collecte / traitement, etc. de ces bibliothèques ou fournir un lien vers leurs politiques de confidentialité (Yu et al., 2016). Un autre problème récurrent provient du caractère évolutif du code de par l'ajout de fonctionnalités par exemple, et du caractère statique des politiques (Slavin et al., 2016). Tous ces facteurs résultent bien souvent en des politiques **incomplètes**, ne couvrant pas tous les accès aux informations sensibles par une application; **incohérentes**, incorporant des conflits internes entre plusieurs déclarations; ou même **erronées**, déclarant ne pas collecter certaines données, qui le sont de facto.

Dans le cadre de notre étude, les applications étudiées sont populaires et imposantes, c'est donc sans crainte que nous pouvons affirmer que leurs politiques sont rédigées par des avocats et non leurs développeurs. Aussi, au vu de leur notoriété, nous pouvons supposer que leurs politiques de confidentialité font office de modèle à des équipes plus modestes du même domaine applicatif. Il est donc essentiel d'identifier leurs problèmes et répercussions possibles. L'analyse est ici réalisée au niveau uniquement des déclarations de collecte de données captées et les aptitudes de contrôle annoncées aux utilisateurs sur cette pratique.

4.2.2 Présentation du jeu de données étudié

Les politiques des dix applications ont été collectées via le lien fourni sur la fiche Google Play de chaque application. Nous avons préféré étudier les textes en anglais, car certaines incohérences et problèmes du langage naturel peuvent être introduits lors de la traduction.

Dans un premier temps, nous avons analysé les politiques de confidentialité dans leur globalité et les avons comparées structurellement. Pour l'analyse du contenu, nous nous sommes concentrés uniquement sur les informations propres à l'étude (*Voir* tableau 4.1).

Tableau 4-1 Structure du jeu de données étudié

	Facebook et Messenger	**Youtube**	**Whatsapp**	**Instagram**	**Twitter**	**Skype**	**Snapchat**	**Pinterest**	**LinkedIn**
Nombre de pages	7	9	10	4	10	14	8	8	8
Nombre de mots des sections de collecte et choix	742	928	1179	1041	2401	3036	1403	1168	1425

La date de dernière révision de la politique est une donnée clé, car elle constitue un indicateur de cohérence entre politique et pratiques actuelles concernant les données utilisateur. Toutes les politiques étudiées indiquent cette date, à l'exception d'Instagram et de Pinterest qui indiquent seulement la date d'entrée en vigueur de leurs politiques (respectivement le 19 janvier 2013 et le 1er novembre 2016). Pour WhatsApp, Facebook et Messenger les politiques n'ont pas été revues depuis 2016, Twitter depuis juin 2017 enfin YouTube, Skype et Snapchat ont des politiques mises à jour assez récemment (entre décembre 2017 et mars 2018). Aussi, toutes ces politiques sont en cours de modification suite au *Règlement Général sur la Protection des Données* (RGPD) applicable dans l'ensemble de l'Union Européenne à compter du 25 mai 2018.

Structurellement, l'ensemble des politiques étudiées est assez homogène : elles sont toutes relativement longues, entre 2300 et 5000 mots environ, articulées en plusieurs sections, entre 6 et 12, avec un nombre important de liens hypertexte, entre 10 et 100. D'autre part, aucune n'est propre à l'application, la politique de confidentialité s'applique à l'ensemble des services de la société mère. Les applications Facebook et Messenger ont ainsi la même politique de confidentialité. Notons ici qu'une mise à jour post-étude des politiques de confidentialité a intégré Instagram dans la politique commune des Services Facebook. Les applications YouTube et Skype se démarquent, car elles appartiennent respectivement à Google et à Microsoft, des géants ayant des services variés dans différents secteurs d'activité. Les politiques de confidentialité de Google citent de manière éparse le service YouTube (plateforme Web et application confondue), pour illustrer des notions abordées. Microsoft fait de même avec Skype, mais lui consacre également une sous-section spécifique. Aussi, de par ce choix de présentation, la politique de confidentialité de Microsoft a une structure qui sort du lot avec près de 25000 mots et plus de 50 sections dont 40 sont des sections dédiées pour chaque service.

Nous pouvons souligner les efforts de LinkedIn et Microsoft en termes de présentation. LinkedIn est la seule entreprise à proposer une courte vidéo pour présenter sa politique de confidentialité. Sur leur support textuel, une petite synthèse à chaque début de section permet

de mieux comprendre quelles informations seront traitées dans la section. Microsoft se base sur un principe similaire : le document textuel présenté initialement consiste en un ensemble de résumés pour ne pas accabler le lecteur qui pourra choisir par la suite « d'en savoir plus », et ainsi d'avoir un complément plus détaillé des sections qui l'intéressent.

Facebook fait également des efforts de présentation en utilisant des couleurs de cadre pour bien délimiter les sections. WhatsApp présente sa politique comme partie intégrante de ses « Mentions légales », ce qui peut fausser la perception de complexité et longueur du document. Néanmoins, tout comme Facebook, Instagram, Snapchat et Pinterest, WhatsApp présente un document sobre et aéré où le corps des sections est développé sous forme de listes en items.

Google a une présentation plus dense, par l'abondance d'hyperliens et de notions complexes soulignées en pointillés pointant vers une définition ou des exemples supplémentaires. L'utilisateur peut vite être submergé par la profusion d'information, mais le texte reste aéré. La politique de confidentialité de Twitter a, quant à elle, une présentation rebutant la lecture de par son caractère dense avec des délimitations de sections peu marquées, de longs paragraphes sans saut de lignes ou items et une quantité impressionnante d'hyperliens.

Toutes les politiques ont été consultées sur le mobile de test afin de constater les difficultés qu'un utilisateur peut rencontrer s'il recherche exactement l'information nécessaire, à savoir quelles informations captées sont collectées par l'application et quels sont ses moyens de contrôle sur une telle pratique. Nous avons constaté un léger gain de temps pour identifier l'information utile quand la politique incluait un index de ses sections, ce qui était le cas pour quatre d'entre elles : Facebook, Google, Microsoft et LinkedIn. Notons ici qu'un index permet de plus une meilleure appréhension, pour le lecteur, de l'ensemble des informations communiquées dans le document. Toutes les politiques ont une section dédiée aux données collectées, mais la distinction entre données sciemment saisies par l'utilisateur et données collectées automatiquement est rarement explicite. D'autre part, toutes les politiques étudiées jouissent d'une section dédiée aux choix en termes de contrôle de données utilisateur, mais

bien souvent ces choix n'y sont que vaguement énoncés ou accessibles via des hyperliens type « En savoir plus ». Aussi, sans lire dans le détail et suivre les liens il est impossible pour l'utilisateur de savoir, a priori où se trouve l'information recherchée : quels sont les moyens de contrôle sur la collecte des données captées.

4.2.3 Déclarations de collecte de données captées et choix utilisateur

Il est question dans cette section de comparer les pratiques annoncées de collectes de données captées et les choix utilisateur correspondant des neuf politiques de confidentialité. Comme précisé précédemment, toutes les politiques ont une section dédiée à la collecte des données et une au choix utilisateur, néanmoins seules celles de WhatsApp, Snapchat, Google et Pinterest font une distinction claire entre les données saisies par l'utilisateur, les données captées et les données communiquées par des tiers. LinkedIn propose une distinction semblable entre données saisies et fournies par des tiers, mais les données captées sont éparses dans plusieurs autres catégories telles que « Utilisation des services », « Votre appareil et votre position », « Messages », etc. Instagram et Facebook ont une catégorisation des données collectées vague centrée sur les actions des utilisateurs. Enfin, Twitter est ici un peu à part, car sa politique traite conjointement collecte et utilisation de données dans une seule et même section composée de nombreux paragraphes disparates en longueur. Il n'est pas possible d'identifier la distinction entre données captées, saisies et fournies sans lire au moins les premières phrases de chaque paragraphe.

Notre méthodologie se base sur des études construisant des *ontologies*[1] afin d'automatiser l'analyse d'un ensemble conséquent de politiques de confidentialité de sites et applications mobiles (Oltramari et al., 2017; Slavin et al., 2016). Notre cadre d'étude est bien plus restreint, de par le nombre de politiques étudiées et le domaine commun des applications auxquelles elles s'appliquent. De plus, leur analyse porte sur la totalité du document, nous

[1] Modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que des relations entre ces concepts.

nous restreignons aux sections concernant la collecte des données. Nous avons donc bâti une pseudo-ontologie pour les données captées et les choix utilisateur associés.

Nous avons dans un premier temps identifié les différentes expressions utilisées pour décrire les informations captées. Nous avons annoté les politiques en identifiant les expressions générales telles qu'« informations techniques » ou « données de connexion », les termes spécifiques tels qu'« adresse IP » « identifiant d'application », et les figures de style telles que les périphrases remplaçant un mot par sa définition ou encore les métonymies permettant d'illustrer un concept par un exemple « localisation d'une photo » pour « métadonnées ». Nous avons ainsi pu étudier les relations hiérarchiques et transitives du langage naturel : « l'adresse IP » est un type de « données de connexion », l'expression «identificateur d'application» est spécifique, mais son concept est couvert par l'expression «informations techniques». Nous avons, par la suite, défini des concepts génériques de données captées et extrait les expressions des différentes politiques se référant aux mêmes concepts. Ce processus d'analyse a abouti à une collection de concepts qui codifient les différentes catégories de données captées, leurs attributs descriptifs, précisions et définitions, et les valeurs d'attributs, à savoir des exemples, telles qu'elles sont dans les neuf politiques de confidentialité.

Notre pseudo-ontologie comporte sept catégories de données captées :

1. **les métadonnées liées au contenu utilisateur** : le contenu utilisateur désigne ici l'ensemble des données sciemment saisies ou partagées par l'utilisateur sur l'application. Les métadonnées constituent donc toute information décrivant ce contenu, date de publication, type de données (texte, multimédia, etc.), localisation, etc. Dans le cas d'application de messageries instantanée, la notion de contenu est plus subtile et se rapporte à toutes les informations « publiquement visibles » à savoir les informations relatives au compte / profil (message, photo, etc.) et les données de statut (en ligne, dernière connexion à … , etc.);
2. **les données personnelles synchronisées** : englobe toutes les données utilisateur des applications intégrées pouvant être automatiquement récupérées et synchronisées avec

les applications téléchargées telles que les contacts, le calendrier, le journal d'appel, les messages texte, les courriels, etc.;

3. **le contenu des communications intégrées** : si l'application incorpore une fonctionnalité de messagerie privée, cette catégorie explicite comment le corps des messages échangés sont considérés et collectés;
4. **les métadonnées des communications intégrées** : ici encore, s'il y a une messagerie privée, cette catégorie explicite quelles informations relatives aux messages échangés sont collectées (date, heure, interlocuteurs, etc.);
5. **les données d'utilisation de l'application** : cette catégorie englobe toute information générée relative au service, à savoir la journalisation des consultations de contenu et autres interactions avec le service (connexion, clic, vue, etc.), les données de performances et d'erreur;
6. **les données de l'appareil et de connexion** : cette catégorie, comme son nom l'indique, fait référence aux données spécifiques du mobile (identifiants, modèle, données de configurations, système d'exploitation, etc.) et de connexion (adresse IP, opérateur et réseau mobile, numéro de téléphone, etc.);
7. **les données de localisation** : ici aussi, le nom est assez explicite, cette catégorie recense les informations collectées concernant l'emplacement physique du mobile.

Dans le tableau 4.2 nous exposons, pour chaque politique, quelle catégorie de données captées sur le mobile est clairement énoncée (en bleu), non informée (NI en rouge), non explicite (NE en orange) ou incomplète (I en jaune). Selon l'application à laquelle la politique se réfère, certaines catégories sont non applicables (NA en gris). C'est le cas par exemple de YouTube qui n'intègre pas de fonctionnalité de messagerie instantanée et donc ne peut collecter des données et métadonnées des messages échangés. À l'inverse, Pinterest et Instagram ont une telle fonctionnalité, mais ne le stipulent pas dans leur politique (NI). Notons, tout de même qu'il est possible, comme le formule Facebook, de considérer les communications privées, via la messagerie instantanée, comme du contenu utilisateur et dans ce cas le problème ne se pose pas. Nous avons considéré ici que la collecte de localisation n'était pas indiquée pour Instagram (NI), bien que le terme localisation soit présent dans sa

politique de confidentialité. En effet, l'exemple de la fonctionnalité du géotag pour expliquer le principe de métadonnée sciemment ajoutée par l'utilisateur (localisation précise d'une photo par exemple) ne couvre en aucun cas la collecte des données de localisation.

Tableau 4-2 Collecte annoncée des catégories de données captées

	Facebook et Messenger	**Youtube**	**Whatsapp**	**Instagram**	**Twitter**	**Skype**	**Snapchat**	**Pinterest**	**LinkedIn**
Métadonnées du contenu utilisateur		NE				NE		NE	NE
Données synchronisées	I	NI				I		NE	
Contenu des communications		NA		NI				NI	
Métadonnées des communications		NA		NI				NI	
Données d'utilisation									
Données du mobile et de connexion									
Données de localisation				NI					

Légende

	Clairement Énoncé	I	Incomplet	NI	Non Informé
NA	Non Applicable	NE	Non Explicite		

Il est important de souligner qu'on peut considérer comme une tâche objective le fait de déterminer si une catégorie est explicitement citée ou non dans une politique, en identifiant les expressions qui se rapportent au concept de la catégorie. Il est déjà plus subjectif de juger de la complétude ou du fait qu'une catégorie soit implicitement citée.

Dans le cas des métadonnées liées au contenu utilisateur les politiques précisent bien souvent qu'elles collectent « les données que vous nous fournissez », cela implique les données sciemment saisies par l'utilisateur, à savoir le contenu et données du compte, **et** leurs métadonnées. D'un point de vue légal, il est clair que la politique n'est pas attaquable, cependant l'utilisateur n'est probablement pas au fait de ce sous-entendu et n'appréhende pas forcément le concept de métadonnée.

On retrouve le même schéma pour la politique de Pinterest qui déclare collecter les « informations que votre appareil met à notre disposition » et les données que «vous nous donnez la permission d'obtenir » ce qui inclut implicitement les données synchronisées telles que le carnet d'adresses du mobile. Pour les politiques plus explicites, cette catégorie est également problématique, car trop souvent limitée uniquement au carnet d'adresses ce qui dans le cas de Facebook, par exemple, est incomplet : de fait, les applications Facebook et Messenger synchronisent également, avec le consentement de l'utilisateur, les messages standards (SMS/MMS) et le calendrier. Il en est de même pour Skype qui collecte les messages standards sur le mobile. Le cas de la politique de WhatsApp est un peu à part, car celle-ci informe l'utilisateur qu'il fournit les données de son carnet d'adresses (contacts) et qu'il confirme être autorisé à le faire. L'utilisateur est présenté comme acteur et l'on peut s'interroger sur le caractère saisi ou capté des données. Comme il est question ici de perception et que la synchronisation est automatique, nous considèrerons qu'il s'agit de données captées.

Les données de contenu et les informations relatives aux communications échangées via les messageries instantanées intégrées de l'application sont sujettes à interprétation. Comme déjà cité, il est possible de considérer ces messages comme un contenu utilisateur, en effet ce sont bien des « données que vous envoyez via nos services ». Cependant, de par leur caractère non public, il est probable que les utilisateurs considèrent ces messages comme privés et donc sensibles. Il est donc essentiel de bien définir le cadre de collecte de ces données. Ainsi, même si bien souvent les politiques qualifient les messages de données privées, leur contenu et leurs métadonnées sont collectés. WhatsApp se démarque en proposant un service, par

défaut, de chiffrement bout en bout empêchant à tout service tiers (WhatsApp compris) de collecter le contenu des messages.

Les informations d'utilisation et les données du mobile et de connexion sont correctement exposées par l'ensemble des politiques avec des écarts importants en termes de présentation. La difficulté réside ici dans le fait que ces données sont plutôt techniques et peu connues du grand public telles que les identifiants d'appareil, d'application de publicité, etc. Par analogie, on peut dire que ces données sont l'équivalent actuel, sur mobile, des témoins de connexion (*cookies*) sur les navigateurs. Les politiques d'Instagram et Pinterest sont très sommaires, mais définissent dans les grandes lignes les informations de l'appareil et les « données de log » : « données techniques envoyées par votre mobile et/ou nos services lorsque vous les utilisés » qui regroupent données d'utilisation et de connexion dans une seule expression. La politique de Facebook associe les données d'utilisation aux informations fournies par l'utilisateur et les données de connexion aux données de l'appareil. L'explication est claire avec des exemples pour illustrer des concepts assez vagues. Twitter et WhatsApp ont le même niveau de langage et détails que Facebook, mais regroupent les concepts différemment : WhatsApp fait deux sections une pour les données d'utilisation, une pour celles de l'appareil et connexion alors que Twitter ne fait qu'une seule et même section pour l'ensemble. Les sections de YouTube et Skype sont particulièrement détaillées et catégorisent en plusieurs sous concepts chaque type de données techniques. Ici aussi, des exemples représentatifs aident fortement à la compréhension. Les politiques de Snapchat et LinkedIn sont moins verbeuses, mais tout aussi claires, probablement par le choix d'exemples illustratifs propre au service.

La collecte des données de localisation est explicitement énoncée dans l'ensemble des politiques, hormis pour le cas déjà cité d'Instagram. Les formulations sont diverses et variées, mais ont toutes le même sens : les données de localisation peuvent être recueillies avec le consentement utilisateur (autorisation sur mobile) et sont nécessaires pour les fonctionnalités avancées de localisation. Ces données sont déterminées précisément grâce aux informations du mobile telles que des coordonnées GPS, les informations sur les réseaux

sans fil, les antennes relais ou les points d'accès Wi-Fi à proximité de l'appareil ou moins précisément à partir de l'adresse IP de l'appareil. Snapchat déclare utiliser également les données des capteurs tels que gyroscopes, accéléromètres et boussoles pour déterminer précisément la localisation de l'appareil.

Comme le souligne l'étude de Sathyendra, Wilson, Schaub, Zimmeck, et Sadeh (2017), l'information la plus utile et exploitable par l'utilisateur est celle concernant les choix et contrôle possibles. Les sections propres à ces informations de toutes les politiques étudiées sont bien souvent peu, voire pas détaillées, et souvent ne fournissent aucun choix relatif à la collecte. Par exemple, Google, précise « Notre objectif est d'être clair sur les informations que nous recueillons, afin que vous puissiez faire des choix informés sur la façon dont elles sont utilisées » (Google LLC, 2018); le choix se rapporte donc uniquement au traitement et non à la collecte. En général, la section dédiée aux choix explique notamment la possibilité de consulter et modifier les informations associées au compte qui sont stockées sur les serveurs du fournisseur de service via des « tableaux de bord » (Facebook, Google, Microsoft), ou les paramètres de l'application / du profil (Twitter, WhatsApp, Instagram, LinkedIn, Pinterest, Snapchat), ou encore comment supprimer le compte. Notons toutefois que les politiques de Snapchat et Pinterest rappellent, dans leur section choix, que l'utilisateur peut utiliser le système d'autorisations de leur plateforme mobile afin de contrôler l'accès aux ressources de l'appareil et donc la collecte de leurs données (synchronisées et de localisation).

Il est courant que les choix soient évoqués à la volée dans les sections de collecte, traitement, stockage, etc., et noyés parmi d'autres informations. Nous avons tiré profit de la syntaxe et du lexique spécifique à la présentation d'un choix afin d'extraire ces déclarations à la volée. Nous avons relevé plusieurs faits notables :

- La politique de Twitter abonde de choix : de fait, chaque paragraphe du document traite un type de donnée (Information du compte, informations publiques, messages, information de localisation, etc.) et précise comment elles sont collectées, traitées et les moyens de contrôle mis à disposition de l'utilisateur. Ces moyens de contrôle sont

accessibles via des liens hypertextes et permettent de paramétrer notamment les préférences de visibilité et collecte concernant la localisation, les contacts et les métadonnées de contenu (liées au tweet) de l'utilisateur.

- Certaines politiques précisent quand la collecte d'un type de données captées est soumise à l'aval de l'utilisateur. C'est par exemple le cas de Facebook pour les données de l'appareil et connexion (incluant la localisation), de Skype pour le carnet d'adresses, de LinkedIn pour la localisation et les cas déjà cités de Snapchat et Pinterest pour la localisation et les contacts.
- La politique de LinkedIn présente la collecte des données synchronisées (contacts, agenda et mails) comme un choix utilisateur pour améliorer et personnaliser le service qui lui est rendu.
- D'autres politiques présentent la collecte comme nécessaire pour fournir des fonctionnalités spécifiques. Il est donc explicitement signalé dans la politique que si l'utilisateur refuse de fournir les données nécessaires, il ne sera pas en mesure d'utiliser la fonctionnalité. C'est le cas de WhatsApp et Skype pour le service de partage de localisation, Instagram et Twitter pour le service d'ajout de métadonnées type géotag au contenu qui implique donc de collecter les données de localisation ou encore pour le service de recherche de contact via le carnet d'adresses du mobile. Notons ici que, dans ce cas, si les données ne sont collectées que dans le cadre de la fonctionnalité citée et que cette dernière est intentionnellement déclenchée et contrôlée par l'utilisateur, nous pourrons considérer qu'il n'y a pas défaut d'intégrité contextuelle.

4.2.4 Obstacles à un consentement averti

Un obstacle majeur à la compréhension et à l'analyse des politiques de confidentialité est qu'il n'y a pas de format canonique pour présenter l'information. La langue, l'organisation et le détail des stratégies peuvent varier d'une politique à l'autre. En plus de ce problème de standard, les politiques étudiées s'appliquent à un ou plusieurs services sur des supports variés (application mobile, navigateur Web), et sont donc bien souvent trop vagues et inclusives. Cela peut induire en erreur et biaiser la perception de l'utilisateur. Par la

formulation suivante « Lorsque vous utilisez les services de Google, nous pouvons recueillir et traiter de l'information sur votre position réelle » (Google LLC, 2018), l'utilisateur peut s'attendre à ce que l'application YouTube accède à sa localisation, cependant s'il se réfère aux autorisations requises la localisation n'y figure pas. Cela permet de souligner un troisième problème, à savoir les incohérences entre autorisation et politique. Dans le cas cité précédemment nous ne pouvons pas à proprement parler d'incohérence, car la collecte annoncée des données de localisation est modulée pas le verbe pouvoir. Enfin, en réponse aux antécédents avec des régulateurs tels que la *Federal Trade Commission* (FTC), résultant dans des mesures coûteuses les entreprises ont recours à des politiques toujours plus inclusives pour ne pas tomber dans les mêmes travers. Par exemple, Snapchat s'est vu imposé 20 ans d'audits en matière de confidentialité et sécurité à cause de leur politique de confidentialité qui déclarait, à tort, ne pas accéder aux données de localisation.

4.3 Autorisations et Permissions

4.3.1 Observations à trois niveaux : Google Play, paramètres et manifeste

Comme déjà mentionné dans le premier chapitre, les autorisations ne sont pas explicitement communiquées à l'utilisateur. Les autorisations normales sont accordées d'office et les dangereuses sont approuvées par groupe. Cependant, un utilisateur soucieux du contrôle de ses données personnelles peut consulter la liste exhaustive des autorisations demandées pour chaque application téléchargée. Cette liste, principal indicateur des pratiques en termes de collecte de données captées, est consultable sur la fiche Google Play de l'application en question et dans les paramètres Android (chemin : Paramètres/Applications/*Application concernée*/Autorisations/Menu/Toutes les autorisations). Rappelons que l'autorisation n'est qu'une traduction en langage naturel d'une permission d'accès à une ressource requise par une application. La liste des permissions requises est quant à elle déclarée dans le manifeste de l'application.

Sur Google Play les autorisations sont classées en différentes catégories : une catégorie par groupe d'autorisations dangereuses (*Agenda, Appareil photo, Contacts, Localisation,*

Microphone, Téléphone, Capteurs corporels, Messagerie texte, Stockage) et une catégorie « Autres ». Notons ici que la documentation officielle d'Aide Google Play (2017) précise que les autorisations présentées varient selon les appareils. En effet, nous avons pu constater que la liste consultée sur le mobile de test différait de celle consultée sur un ordinateur. Cela s'explique par le fait que les applications sont destinées à des plateformes mobiles et la liste affichée sur l'ordinateur englobe les autorisations requises de plusieurs versions de l'application et pas forcément de celle disponible sur le mobile.

Dans les paramètres du mobile, les autorisations sont classées comme sur Google Play, mais le nombre d'autorisations dans la catégorie « Autres » est systématiquement inférieur. Cependant, à y regarder de plus près, les autorisations différant de l'affichage Google Play sont présentées dans les informations de l'application en tant que paramètres avancés et sont révocables. Aussi en cliquant sur les autorisations individuellement, l'utilisateur pourra avoir un petit texte explicatif de son utilité.

Dans le manifeste, une permission d'accès requise par l'application est déclarée dans une balise < *uses-permission* >. Nous avons donc récupéré le manifeste de chaque application et prélevé l'ensemble des déclarations < *uses-permission* >. Nous avons identifié deux catégories de permissions : les standards, définies par Android, et les autres. La seconde catégorie peut être subdivisée en trois sous-catégories : les permissions déclarées par l'application elle-même, celles définies par un constructeur d'appareil ou un lanceur d'applications (*launcher*), et celles exigées par des applications ou des bibliothèques tierces (*Voir* ANNEXE I). Cette catégorisation se base sur les dénominations *reverse-DNS* des permissions (*Voir* section 1.2.3.1) ainsi que sur les balises < *permission* > présentes dans le manifeste. Comme déjà vu dans les notions préliminaires, ces balises comportent la déclaration même des permissions propres à l'application. Enfin, comme la balise < *uses-permission* > ne contient que l'attribut *name*, nous avons dû nous reporter à de la documentation complémentaire (documentation Android, Google Play, manifestes d'applications tierces, etc.) afin de renseigner les différentes caractéristiques des permissions, à savoir leur niveau de protection et leur groupe s'il y a lieu.

4.3.2 Défaut d'information et explications

Après la collecte et la catégorisation de ces informations, nous avons associé chaque permission à une autorisation pour pouvoir identifier les permissions non déclarées à l'utilisateur et comprendre la cause de ce défaut d'information. Nous présentons les résultats et analyses selon le niveau de protection des permissions.

4.3.2.1 Les permissions / autorisations dangereuses

Dans le tableau 4.3 nous reportons le nombre de permissions dangereuses requises dans le manifeste, le nombre d'autorisations correspondantes correctement informées à l'utilisateur dans les paramètres, et l'écart entre les deux. En cas d'écart, nous avons donc examiné les permissions en question et identifié la cause du **défaut d'information** (écart négatif) ou sur-demande (écart positif).

Tableau 4-3 Comparaison permissions / autorisations dangereuses

	Facebook	**Messenger**	**Youtube**	**Whatsapp**	**Instagram**	**Twitter**	**Skype**	**Snapchat**	**Pinterest**	**LinkedIn**
Autorisations	13	19	4	12	10	11	14	9	6	9
Permissions	13	20	3	12	9	11	14	9	6	8
Écart	0	-1	1	0	1	0	0	0	0	1

Ici, Messenger demande une permission dangereuse non annoncée à l'utilisateur, cela est cependant justifié, car la permission en question a été retirée de la version 7 d'Android. Les applications Android sont développées pour une version du cadriciel Android spécifique précisée dans leur manifeste par l'attribut < *android : targetSdk* >. Cependant elles peuvent être déployées sur des appareils ayant des versions du cadriciel Android plus anciennes. Cette rétrocompatibilité, également informée dans le manifeste par l'attribut < *android : minSdk* >, a des conséquences sur les permissions déclarées. Chaque nouvelle version Android introduit

potentiellement de nouvelles permissions et en déprécie et/ou supprime d'autres. Les permissions supprimées sont néanmoins toujours nécessaires sur les appareils utilisant d'anciennes versions de la plateforme. Comme la version du mobile de test est la version 7.0, toutes les permissions dépréciées ou retirées pour cette version, présentes dans le manifeste pour des raisons de rétrocompatibilité, ne sont pas présentées à l'utilisateur.

YouTube, Instagram et LinkedIn requièrent une permission dangereuse de moins que d'autorisations annoncées. Pour les trois applications l'autorisation en sus est la même : « Lire le contenu de la carte SD » qui correspond à la permission standard *« READ_EXTERNAL_STORAGE »*. Ici la différence s'explique par le fait que les trois applications demandent les droits en écriture via la permission *« WRITE_EXTERNAL _STORAGE »* et lors de la compilation le système ajoute automatiquement les droits en lecture. Les autres applications de l'étude demandent explicitement ces deux permissions et ne sont donc pas sujettes à cet ajout automatique.

4.3.2.2 Les permissions / autorisations normales

Dans le tableau 4.4 nous reportons le nombre de permissions normales requises dans le manifeste selon leurs catégories, le nombre d'autorisations correspondantes correctement informées à l'utilisateur et l'écart entre les deux.

Tableau 4-4 Comparaison permissions / autorisations normales par catégorie

	Facebook	Messenger	Youtube	Whatsapp	Instagram	Twitter	Skype	Snapchat	Pinterest	LinkedIn
Autorisations (applicatives)	3	2	3	2	1	2	2	1	2	1
Permissions propres à l'application	6	6	1	5	2	4	2	2	1	0
Permissions applicatives	6	5	3	3	4	2	2	2	2	1
Autorisation (constructeurs)	0	0	0	0	0	0	0	0	0	0
Permissions constructeurs	7	8	0	9	5	2	9	7	2	12
Autorisations (standard)	17	15	8	17	11	10	15	10	4	10
Permissions standard	20	18	10	21	13	14	19	12	6	12
Total autorisations	21	18	11	19	12	12	17	11	6	11
Total permissions	39	37	14	38	24	22	32	23	11	24
Écart	-18	-19	-3	-19	-12	-10	-15	-12	-5	-13

Le cas des permissions normales nécessite de plus amples explications concernant la déclaration des permissions. Comme expliqué dans les notions préliminaires, Android permet aux développeurs et aux constructeurs de créer des ressources et définir des permissions pour les protéger. Pour les développeurs d'applications, ces permissions doivent être définies dans le manifeste comme présenté en figure 4.1.

```
<permission
      android:name="string"
      android:label="string resource"
      android: description="string resource"
      android:icon="drawable resource"
      android:permissionGroup="string"
      android:protectionLevel=["normal" | "dangerous" | "signature" | … ]/>
```

Figure 4.1 Syntaxe de déclaration de permission dans un manifeste Android

Le nom de la permission est spécifié dans l'attribut *name* : il sera utilisé dans le code pour faire référence à cette permission. C'est également ce nom qui sera retenu par le système pour appliquer le contrôle d'accès. L'attribut *permissionGroup* permet d'assigner la permission à un groupe, s'il n'est pas renseigné la permission sera sans groupe. Le niveau de protection de la permission est spécifié dans l'attribut *protectionLevel* et définit comment la permission en elle-même sera accordée à une application la réclamant. Les attributs en bleu

constituent le versant utilisateur de la permission, à savoir sa traduction en langage naturel (*label*) alias l'autorisation correspondante, une description plus détaillée de l'utilité de la permission (*description*), et une icône représentant la permission (*icon*). Ces attributs sont facultatifs et bien souvent non informés.

Twitter est, dans l'ensemble des applications étudiées, la seule à détailler attributs destinés à l'utilisateur pour l'une de ses quatre permissions propres : *« com.twitter.android.permission.AUTH_APP »*. En effet, la permission a un niveau de protection dangereux, ce qui nécessite un consentement explicite de l'utilisateur et donc de figurer dans les autorisations révocables présentes dans les paramètres de l'application. Toutes les autres applications ne déclarent que des permissions normales, à l'exception de Skype, et ne renseignent pas l'attribut *label*. C'est pourquoi aucune des permissions normales propres à l'application n'a d'autorisation correspondante. Notons ici que bien qu'un niveau de protection normal n'implique pas de consentement utilisateur explicite, la permission devrait être mentionnée dans la liste des autorisations consultables, à titre informatif.

Nous pouvons extrapoler ce raisonnement aux permissions déclarées par des applications et bibliothèques tierces. Dans notre jeu de données recueillies, toutes les permissions de cette catégorie à l'exception des quatre permissions définies par les services Google n'ont pas d'autorisation correspondante.

Les permissions constructeur et des lanceurs d'applications ne sont présentées que si l'utilisateur est concerné. De fait, lorsque la permission est destinée à protéger des ressources d'un constructeur X, informer l'utilisateur d'un appareil du constructeur Y est vain et peut même être source d'inquiétude et incompréhension. Le constructeur de notre appareil de test ne figurant pas parmi ceux exigeants les permissions utilisées par les dix applications étudiées, aucune permission constructeur n'a, dans notre jeu de données, d'autorisation correspondante.

Enfin, l'écart concernant les permissions / autorisations standards normales s'explique, comme pour les permissions dangereuses, par le fait que les permissions sont retirées de la version 7.0 d'Android, mais figurent toujours dans les manifestes des applications pour des raisons de rétrocompatibilité.

4.3.2.3 Les permissions signature et les autorisations spéciales

La documentation Android sur les permissions (Android Developpers : Permissions Overview, 2018), bien qu'assez détaillée, reste assez vague sur le cas des permissions signature. L'utilisateur n'a que deux niveaux d'autorisations (dangereux et normal) ainsi les permissions signature n'ont pas de correspondance adéquate. Dès lors, il y a trois possibilités : soit l'autorisation correspondante apparaît dans la catégorie « Autres » et est alors assimilable, à tort, à une permission normale, soit elle est présentée comme un paramètre avancé de l'application, soit, et ce cas est plus problématique, elle n'apparaît pas du tout. Quand la permission est « traduite » en paramètre avancé, elle est consultable, expliquée et révocable. Nous reportons dans le tableau 4.5 les permissions signatures requises par les applications étudiées et leur « traduction », s'il y a lieu. Aussi si une application ne figure pas dans le tableau c'est qu'elle ne demande aucune permission signature.

Tableau 4-5 Cas des permissions signature

Permission signature	Facebook	Messenger	Youtube	Whatsapp	Twitter	Skype	Traduction pour l'utilisateur	Valeur
BATTERY_STATS	X	X					Aucune	
MANAGE_DOCUMENTS			X				Aucune	
SYSTEM_ALERT_WINDOW	X	X			X	X	Paramètre avancé	Afficher par dessus d'autres applications
WRITE_SETTINGS	X			X		X	Paramètre avancé	Modifier les paramètres systèmes
REQUEST_INSTALL_PACKAGES	X	X					Autorisation "Autre"	Demander l'installation de paquets

Notons de plus que Skype déclare deux permissions propres et leur assigne le niveau de protection signature, mais leur attribut *label* n'est pas informé. Ainsi, comme pour le cas des permissions propres de niveau normal, ces permissions ne sont pas présentées à l'utilisateur.

Malgré des recherches poussées dans la documentation Android, des forums et recueils spécialisés, nous n'avons pas pu trouver d'explications à la confusion relative aux permissions signature standard.

4.3.3 Les limitations

Les limitations en termes d'information à l'utilisateur liées au système de notifications et consentement Android sont de deux types : celles imputables aux développeurs d'applications et celles imputables à Android. Pour le cas des développeurs, nous ne pouvons pas leur reprocher de demander des permissions mal déclarées par des applications tierces, à savoir sans label ni description. Par contre, il est raisonnable de les blâmer pour leurs déclarations non détaillées et donc non consultables par l'utilisateur. Pour Android, le problème est principalement lié à la non-existence d'une catégorie d'autorisations spécifiques correspondantes aux permissions signature. Aussi la dépréciation / suppression de permissions devrait être mieux documentée et expliquée : pourquoi la permission est-elle supprimée, est-elle remplacée, la ressource anciennement protégée est-elle en libre accès, etc. Enfin, le modèle donne trop de liberté en termes d'information utilisateur. Il est actuellement possible que des applications créent des ressources et les protègent avec des permissions normales, à savoir attribuées d'office. Dès lors, il suffit, pour une application tierce, de connaître le nom de la permission pour avoir accès à la ressource. Maintenant, considérons que cette ressource est un fournisseur de contenu (bases de données dédiées) qui stocke des données utilisateur telles qu'une liste de contact. Le système actuel de déclaration / demande de permissions permet de faire transiter ces données de contacts d'une application à une autre sur le mobile à l'insu de l'utilisateur. Dans notre jeu de permissions, nous avons pu observer que Facebook et Messenger utilisent un tel mécanisme. Dans une optique de transparence et d'éthique, il serait raisonnable qu'Android rende l'attribut *label* de la balise < *permission* >

obligatoire, quel que soit le niveau de protection afin de mieux informer un utilisateur soucieux de sa vie privée.

La cohérence entre les autorisations et les politiques de confidentialité est difficile à évaluer. Les autorisations / permissions constituent le versant technique du processus de notification et consentement et sont donc spécifiques et peu significatives pour un utilisateur non spécialisé. Les politiques sont, elles, le versant juridique et sont donc plus vagues et inclusives. Une expression dans la politique peut alors inclure diverses permissions et, à l'inverse, une permission peut être commune à différentes expressions. De fait, techniquement, il est possible de déduire la localisation avec les données de réseaux mobiles et Internet (adresse IP, borne Wi-Fi, antennes relais à proximité) donc les développeurs accédant à ces données sont tenus de demander la permission *« ACCESS_COARSE_ LOCATION »* dans le manifeste. Cependant, dans la politique cela peut être couvert par l'expression « données de connexion » non informatives sur l'aspect localisation. À l'inverse, il est possible que dans une politique, il soit précisé que l'application utilise par exemple des traceurs sans qu'il n'existe de permission / autorisation correspondante, comme par exemple, dans la section « Device identifier » de la politique d'Instagram (Instagram Inc.).

CHAPITRE 5

ÉTUDE DES ACCÈS AUX RESSOURCES

5.1 Cadre de l'étude

L'étude vise à déterminer des statistiques d'accès aux ressources protégées selon le contexte d'exécution afin d'évaluer si les applications ont un « comportement » raisonnable, à savoir en accord avec les attentes utilisateur, ou si elles « abusent » des permissions qui lui sont accordées.

Rappelons ici que pour des raisons de faisabilité, nous avons restreint l'étude pratique aux groupes de permissions *Localisation* et *Messagerie texte*. L'application YouTube ne requiert aucune permission de ces groupes et ne figure donc pas dans l'étude pratique.

Par souci de validité des résultats, nous tentons au maximum de limiter les biais techniques qui pourraient modifier le « comportement » des applications testées. Comme expliqué en section 2.2.3, il est nécessaire de débrider le mobile afin d'utiliser le cadriciel *Xposed* nous permettant de tracer les appels aux méthodes d'accès aux ressources. Un débridage conventionnel modifie le système d'exploitation du mobile et, avec lui, son système de fichiers. Nous craignons que ces modifications puissent affecter le « comportement » des applications. De plus, certaines applications utilisant *SafetyNet*, un service Google permettant de détecter le débridage, cessent de fonctionner. C'est le cas par exemple de Snapchat. C'est pourquoi nous avons utilisé le cadriciel *Magisk Manager* (Wu, J et al., 2016), une solution de débridage alternative ne modifiant pas le système de fichier et non détectable par les applications.

5.2 Interactions inter-applications : Précisions sur les Intents

Nous détaillons ici les mécanismes de communication inter-applications (inter-composants) présentés en section 1.1.1. Pour rappel, l'ICC se base sur l'échange de messages asynchrones, nommés Intents, qui contiennent une tâche à déléguer à un composant tiers, et s'il y a lieu, l'URI des données à traiter.

Si une application X envoie un Intent implicite, l'application Y traitant l'Intent exécutera l'action avec ses permissions d'accès. Dans ce mémoire, nous considérons qu'il y a collecte de données (de localisation ou de messagerie texte) par l'application X même si l'action est réalisée par l'application Y via les mécanismes d'Intent. Il est donc essentiel de tracer les appels et retours d'Intent, et d'identifier dynamiquement le processus appelant et exécutant.

Dans l'écosystème Android, une application doit détenir des permissions précises pour pouvoir invoquer certaines actions prédéfinies dans un Intent implicite. Par exemple *« ACTION_CALL »*, délégant à l'application de téléphone la tâche d'initier un appel, requiert la permission *« CALL_PHONE ».* Cependant, plusieurs actions prédéfinies par Android permettent d'accéder à des ressources sensibles sans permissions. Le contrôle d'accès, via les permissions acquises de l'application appelante, est alors remplacé par une interaction utilisateur. Ainsi, si l'application demandeuse utilise *« ACTION_DIAL »* à la place de *« ACTION_CALL »*, aucune permission n'est requise, mais l'utilisateur doit cliquer sur le bouton « Appel » de l'application téléphone (exécutant l'action) pour initier l'appel. En d'autres termes, l'interaction utilisateur constitue un consentement de facto.

Il y a ici, comme dans le cas du consentement par clic des politiques de confidentialité, un défaut d'informations. Le problème est que certaines actions non protégées permettent un accès à des ressources sensibles du mobile telles que la caméra (*« ACTION_IMAGE_CAPTURE »*, *« ACTION_VIDEO_CAPTURE »*), les fournisseurs de contenu de contacts (*« ACTION_PICK »*, *« ACTION_VIEW »*, *« ACTION_EDIT »*) messagerie texte *(« ACTION_SEND »*), etc. (Android Developpers : Intent Common, 2018), et exposent des données

personnelles à l'ensemble des applications pouvant les traiter. L'utilisateur n'est pas au fait de ces données techniques et n'associe probablement pas son action à un consentement. Il y a donc structurellement un défaut d'intégrité contextuelle dans ce mécanisme d'Intent.

L'outil de trace capture l'ensemble des Intents émis par les applications étudiées, néanmoins, l'objectif de l'étude n'étant pas de faire de la rétro-ingénierie, nous nous concentrons sur ceux concernant la localisation et la messagerie texte cités dans *The Busy Coder's Guide to Android Development 8.9* (Murphy, 2017) que nous avons utilisés comme base de connaissances.

5.3 Outils : collecte et traitement des données

Comme précisé dans la méthodologie, notre outil de trace ***PrivacyStats*** est basé sur le module *Xprivacy* du cadriciel *Xposed* (Rovo89 & Tungstwenty, 2011). Ce cadriciel utilise le mécanisme de démarrage d'Android afin de permettre d'agir sur le contexte d'exécution de tous processus. Plus précisément, il étend l'exécutable qui charge les classes nécessaires et appelle les méthodes d'initialisation du processus « Zygote ». Comme sur Android, toute application est démarrée comme une copie (*fork*) du Zygote, cet exécutable permet de modifier dynamiquement le fonctionnement de l'application. Il est cependant important de comprendre que ce cadriciel ne modifie pas le code source à l'intérieur des méthodes des applications instrumentées. Au lieu de cela, il permet d'injecter du code spécifique avant et après l'appel à une méthode (plus petite unité de Java pouvant être adressée clairement) via la classe XposedBridge (Rovo89, 2017), et ce sans que l'appelant ne le sache. Il est alors possible de changer les arguments pour l'appel, invoquer d'autres méthodes, modifier le résultat, ignorer l'appel, etc.

XPrivacy (Bokhorst, M. 2012) instancie différents XposedBridge afin de modifier un grand nombre de fonctions Android et ainsi limiter les catégories de données sensibles auxquelles une application installée peut accéder. Selon la fonction, *XPrivacy* ignore l'exécution de la

fonction d'origine ou modifie ses arguments ou son résultat afin de remplacer les données sensibles transmises aux applications par des données factices ou vides.

Notre outil de trace, ***PrivacyStats,*** instancie les mêmes XposedBridge sur les fonctions Android relatives aux données de localisation et de messagerie texte, mais ne modifie par leur modèle d'exécution. Pour chaque fonction nous développons un code spécifique, dit un « hook », afin de journaliser l'accès aux ressources de façon adéquate. Cette étape de développement a été assez complexe, car elle impliquait des opérations sensibles. La moindre erreur dans le code de capture d'accès, le hook, ou dans les traces générées, pouvait provoquer un démarrage en boucle (*bootloop*). Parvenir à retrouver une version stable nécessitait d'intervenir au niveau du système et parfois de tout réinstaller, ce qui était évidemment très chronophage.

Le cadriciel *Xposed* a un système de fichiers journaux intégré, mais ce dernier a une limitation de taille. Il a donc été nécessaire d'utiliser le journal système natif d'Android *LogCat* (Android Developpers : Write and View Logs with Logcat, 2018) couplé à l'application *CatLog* qui permet de sauvegarder les logs en temps réel sur la carte SD du mobile. Afin que nos traces soient aisément manipulables, nous avons utilisé un format CSV (*Comma Separated Values*) pour les traces générées. Aussi, pour réaliser des mesures révélatrices du comportement des applications étudiées et comparables entre elles, nous avons établi différents protocoles expérimentaux : applications actives, fonctionnalités à stimuler, activité utilisateur, etc.

Une fois enregistrées sur le mobile, les traces sont importées sur ordinateur et nommées en utilisant un ***script Python***. Un second ***script Python et SQL*** est dédié au stockage des données brutes dans un fichier d'historique et l'ajout incrémental des données filtrées (uniquement les traces générées par notre outil ***PrivacyStats***) dans une ***base de données SQLite***. Il nous a semblé pertinent de tirer profit de la puissance des bases de données relationnelles et de SQLite pour réaliser nos statistiques d'accès. Notons qu'il a été

nécessaire d'avoir une grande rigueur dans la normalisation des traces pour l'extraction de données utiles à insérer dans la base de données.

La base de données a quatre tables principales :

- *Expe :* recense les données de chaque expérimentation (nom, durée, fichier de traces correspondant);
- *Events* : contient les traces d'appels aux méthodes instrumentées et le contexte d'appel (expérience, application appelante, hook sollicité, visibilité de l'application appelante, activité utilisateur, date et heure de l'appel);
- *Errors* : inclut toutes les traces d'erreurs afin de vérifier qu'aucun problème n'affecte la fiabilité des mesures;
- *Intents :* table plus détaillée spécifique aux mécanismes assez complexes d'Intents permettant de recenser le contexte d'appel ainsi que les identifiants des applications émettrices et réceptrices du message, l'action requise, et, le cas échéant, les informations supplémentaires transmises (composant cible, URIs, etc.).

Au cours des expérimentations, certaines traces se sont révélées non significatives, constituant alors des données parasites devant être supprimées. De fait, l'appel à certaines méthodes protégées par une permission du groupe *Localisation* ne constitue pas un accès effectif à la localisation du mobile. C'est le cas, par exemple, de la méthode *isProviderEnabled* qui renvoie l'état (activé / désactivé) du fournisseur passé en paramètre (*GPS, Network, Fused, Passive*) et donc n'accède pas à la localisation à proprement parler. Nous avons introduit différentes tables *IgnoreExpe*, *IgnoreProcess*, *IgnoreHook*, etc. qui permettent de filtrer graduellement et précisément l'ensemble des traces sans avoir à modifier l'outil de capture et refaire l'ensemble des mesures, ce qui aurait été trop coûteux en ressources et en temps. Enfin, nous utilisons le mécanisme de vues pour agréger les données utiles, mettre en évidence certaines corrélations et interpréter les résultats.

5.4 Statistiques d'accès aux ressources

5.4.1 Fonctionnalités de localisation et messagerie texte

Bien que toutes les applications étudiées soient des médias sociaux, elles couvrent quatre catégories distinctes : photographie, messagerie, microblogage et réseautage social. Nous les présentons rapidement en mettant l'accent sur les fonctionnalités en lien avec la localisation et la messagerie texte, ainsi que quelques exemples de modèles conceptuels.

Les applications de photographie, Snapchat et Instagram, permettent aux utilisateurs de prendre des photos et/ou des vidéos, les éditer, puis les partager avec d'autres de manière novatrice (Filtres, Stories : contenu multimédia visible pendant 24 heures, etc.). Snapchat a une fonctionnalité de cartographie qui permet de localiser ses amis à l'échelle mondiale et découvrir des Stories publiques à proximité. Instagram permet d'ajouter un géotag à une publication. Enfin, les deux adaptent le contenu proposé selon la localisation : filtres géolocalisés (propre à un évènement par exemple), publicité, etc. L'utilisateur, sachant cela peut s'attendre, pour Instagram, à un pic de collecte lorsqu'il géomarque une photographie et à des accès occasionnels à la localisation lorsque l'application est active.

Les applications de messagerie : WhatsApp, Skype et Messenger permettent aux utilisateurs d'échanger des messages texte, des images et des vidéos. Ils diffèrent des messages texte / multimédia standards (SMS/MMS), car ils permettent d'utiliser les technologies VoIP (*voice over IP*) à savoir s'appuyer sur les données cellulaires (3G/4G) ou le Wi-Fi plutôt que le réseau cellulaire. En plus d'éviter les frais liés aux messages standards, ces applications gagnent en popularité pour leurs fonctionnalités étendues telles que les notifications de remise de messages, le partage de position ponctuelle (toutes) ou en temps réel (WhatsApp et Messenger) pour faciliter les rencontres. Messenger peut également se substituer à l'application de messagerie par défaut sur le mobile (afin de centraliser les messages sur une seule application). Il est évident que pour la fonctionnalité de partage de localisation en temps réel, le modèle conceptuel de l'utilisateur dépend de ses capacités cognitives : un utilisateur lambda imagine probablement que l'application capte sa localisation en continue

tant qu'elle est active, un utilisateur plus avisé peut, de plus, espérer que la collecte fonction de sa vitesse de déplacement.

Les applications de microblogage : Twitter, Facebook et Pinterest permettent aux utilisateurs de partager rapidement de courts messages ou du contenu multimédia (images et vidéos) à une plus ou moins large audience (selon les préférences de confidentialité). Sur Twitter et Facebook, il est possible de géolocaliser une publication. Twitter propose également un service d'alertes par SMS en cas de crise ou d'urgence contenant les informations sur un évènement en cours. L'utilisateur suppose probablement que l'application ne collecte que les messages provenant des serveurs de Twitter. Pinterest n'a pas de fonctionnalité apparente liée à la localisation. Ainsi la demande d'autorisation du groupe *Localisation* peut être perçue comme intrusive.

Les applications de réseaux sociaux, Facebook et LinkedIn, permettent de relier des personnes physiques ou morales entre elles et constituer un réseau social virtuel dans le but de faciliter la gestion de carrières professionnelles (LinkedIn) ou l'échange de contenu autour de centres d'intérêt (Facebook). Facebook, en sus des fonctionnalités liées à son système de microblogage (géolocaliser une publication), propose la fonctionnalité « Amis à proximité » qui permet de notifier l'utilisateur quand un de ses amis est dans les environs. LinkedIn permet d'utiliser les données GPS précises afin de renseigner le code ZIP dans le profil de l'utilisateur.

Toutes les applications étudiées, à l'exception de Pinterest et LinkedIn, permettent de se connecter au service avec le numéro de mobile comme identifiant et proposent une fonctionnalité de confirmation automatique du numéro de mobile par SMS.

L'objectif de l'étude pratique est d'utiliser les applications en sollicitant ou non les fonctionnalités de localisation et de messagerie et chercher à identifier, le cas échéant, des situations d'accès qui défient les attentes utilisateur.

5.4.2 Complexité de l'analyse des mesures

5.4.2.1 Complexité au niveau quantitatif

Afin d'évaluer le « comportement » des applications, nous considérons le nombre d'appels à des méthodes et Intents accédant à la localisation ou aux messages (SMS/MMS). Néanmoins, nous n'avons pas de maîtrise sur l'occurrence des appels. Si nous optons pour une métrique trop simpliste, par exemple la moyenne (nombre d'appels / temps de l'expérience), les résultats peuvent se révéler non significatifs. De fait, si nous considérons une expérience de 30 secondes et que nous avons 30 accès, cette métrique indiquerait un accès par seconde en moyenne. Cependant, il est tout à fait possible que tous ou la plupart des accès soient réalisés dans un intervalle très court (Ex. : $1/10^{ème}$ de seconde), ou au contraire qu'ils soient effectivement réguliers et répartis sur toute la durée de l'expérience. L'information captée a, dans le premier cas, une faible entropie étant donnée la vitesse de déplacement d'un humain : elle est équivalente, en termes informationnels, à un seul accès. Le second cas permet, lui, de traquer les déplacements d'un utilisateur et, même lorsque celui-ci est arrêté, permet de détecter ce fait.

Si les évènements sont isolés, nous tentons d'identifier les catalyseurs des appels : activité utilisateur, fonctionnalité sollicitée, changement d'état de l'application (retour en premier plan), etc. Si par contre, ils sont répétés, nous devons considérer leurs distributions.

Notre métrique doit donc prendre en compte le nombre d'appels aux méthodes de localisation et messagerie texte ainsi que la répartition de ces appels dans le temps. Pour ce faire, à l'aide de requêtes SQL, nous segmentons nos mesures en intervalles de temps réguliers (Ex. : 10 ms, 100ms, 1s ,10 s). L'idée est ensuite de calculer, toujours à l'aide de requête SQL, pour un intervalle de temps donné, le pourcentage d'intervalles où il y a au moins un appel (requêtes d'agrégats avec *count(distinct(round(TS/taille_intervalle)))* où TS représente le temps écoulé en ms, depuis le début de l'expérience). Ce pourcentage, calculé pour plusieurs intervalles de temps, permet de capturer la **fréquence d'appel**. Cependant, en considérant une application faisant cinq appels par seconde, d'un intervalle de segmentation

d'une seconde est trop grossier et laisserait à penser que la fréquence est un appel par seconde, or, la fréquence réelle est un appel toutes les 200 ms. Cela est dû au fait que nous calculons le pourcentage d'intervalles où il y a **au moins** un appel. Dès lors pour trouver la bonne échelle, il est nécessaire d'itérer le calcul, en partant du plus petit intervalle possible (la milliseconde : fréquence de mesure) et en augmentant progressivement la taille de l'intervalle de segmentation.

5.4.2.2 Complexité au niveau qualitatif

Les accès liés à la messagerie texte sont très encadrés et les APIs et mécanismes d'Intents impliqués sont bien documentés. Il a donc été relativement aisé d'appréhender le comportement des applications relatif à cette ressource. Le cas de la localisation est, quant à lui, bien plus complexe à appréhender, et ce pour plusieurs raisons :

- certaines APIs sont privées : la documentation officielle Android recommande de préférer les APIs de localisation des services Google Play plutôt que celles du cadriciel Android. Bien que certaines méthodes soient publiques, il n'est pas évident de connaître l'ensemble des appels APIs à capturer;
- les mécanismes de mises à jour de localisation et de rappel sont peu documentés et impliquent des Intents;
- les applications ont des fonctionnalités diverses et variées en rapport avec la localisation (*Voir* section 5.4.1);
- il existe différents fournisseurs de localisation : **GPS**, déterminant l'emplacement du mobile via des données satellites; **Network** calculant l'emplacement à partir des données de réseaux mobile et Internet (disponibilité des points d'accès Wi-Fi, proximité des antennes relais); **Fused** et **Passive** renvoyant les données d'emplacements générés par les autres fournisseurs.

Lorsque le fournisseur *Fused* est sollicité, il renvoie la meilleure donnée de localisation disponible, à savoir la plus précise possible. *Passive* est, lui, un fournisseur très particulier, dans le sens où il est utilisé pour recevoir « passivement » des mises à jour de localisation

lorsque d'autres applications ou services les demandent. Donc, si aucune autre application sur le mobile ne reçoit de mises à jour de localisation, l'application utilisant le fournisseur *Passive* ne les obtiendra pas non plus. Les traces de débogage et système nous ont permis de mieux comprendre et analyser les accès à la localisation, notamment concernant le fournisseur de localisation *Passive*. Il nous paraît assez clair qu'en cas de besoin fonctionnel d'une donnée de localisation, le fournisseur n'est pas adapté puisque cette localisation ne sera obtenue que si une autre application interroge de facto un autre fournisseur. Ainsi, on peut considérer que le recours à ce fournisseur est une « capture opportuniste » de la localisation pour un objectif probablement non essentiel à l'application, et est donc structurellement abusif en termes d'intégrité contextuelle. Notons cependant l'intérêt de ce fournisseur qui permet de mutualiser les accès à la localisation.

5.5 Résultats

5.5.1 Paramètres expérimentaux

Comme précisé dans la section 5.3, l'objectif des traces est principalement de réaliser des mesures révélatrices du comportement des applications étudiées et comparables entre elles. L'expérimentation s'est étendue sur cinq semaines et comportait deux phases de tests distinctes : la phase de découverte (trois semaines) et la phase contrôlée (deux semaines). Les mesures ont été réalisées par une personne qui n'était pas familière avec les applications.

Durant la première phase, il s'agissait de maîtriser les applications, affiner les outils de traces et identifier les conditions sollicitant des accès aux ressources localisation et messagerie texte pour chaque application. En itérant les tests et en explorant les traces de façon semi-automatique, en s'aidant de la base de données détaillée dans la section précédente et de l'outil de visualisation *DB Browser for SQLite* (Piacentini et al., 2015), il a été possible de corriger les outils de capture en supprimant les *hooks* qui généraient des traces inintéressantes, de détailler les traces signifiantes en ajoutant d'éventuels paramètres à capter, et d'uniformiser l'ensemble en standardisant les formats de traces. À l'aide de la métrique détaillée en section 5.4.2.1 (pourcentage d'intervalles temps où il y a eu au moins un accès)

et d'observations lors des expériences, nous avons pu nous faire une idée du comportement de chaque application. Évidemment les mesures issues de cette phase exploratoire ne peuvent être utilisées pour l'analyse, car elles sont réalisées dans des contextes variés et avec différentes versions de nos outils de traces. Cependant, elles nous ont guidés pour déterminer un plan expérimental clair pour la phase contrôlée.

La seconde, phase, bien plus systématique a permis de fournir un ensemble de mesures contrôlées sur lesquelles reposent notre analyse et nos résultats. Nous avons fait varier les paramètres d'expérimentation, détaillés dans le tableau 5.1 ci-dessous, et réalisé des mesures courtes, d'environ cinq minutes, dans un contexte précis.

Tableau 5-1 Paramètres expérimentaux

Application(s) testée(s)	**Données observées**	**Activité de l'utilisateur**	**Visibilité de l'application**	**État du mobile**	**Fonctionnalités sollicitées**
Facebook, Messenger, Twitter, Instagram, Snapchat, WhatsApp, Skype, Pinterest, LinkedIn	Localisation Messagerie texte	Statique Marche	Premier plan Arrière-plan	Allumé Veille	Envoi ou réception de SMS Géolocalisation ponctuelle ou continue Amis proches etc. *Voir* section 5.4.1

En nous basant sur les conclusions de la phase exploratoire, nous avons testé les applications en isolation, puis ensemble en faisant varier les paramètres pertinents : par exemple, si nous constatons que Snapchat n'accède à la localisation que lorsqu'elle est au premier plan, nous stoppons la campagne de mesure avec Snapchat en arrière-plan. Cette stratégie permet d'avoir un nombre d'expériences raisonnable.

Pour présenter les résultats nous avons utilisé une métrique similaire à celle développée et utilisée en phase de découverte, mais ayant un rendu graphique plus intéressant : le nombre d'accès par seconde. De fait, nous avons pu observer lors de la phase exploratoire que la plupart des comportements notoires sont mis en exergue en prenant une seconde comme intervalle de temps d'échantillonnage.

Les combinaisons application(s) / contexte (paramètres expérimentaux) les plus révélatrices du comportement de collecte sont reportées dans le tableau 5.2 ci-après. Si, pour un paramètre donné, il est spécifié « indifférent », cela signifie que le comportement est similaire, quel que soit la valeur employée. Nous indiquons aussi dans ce tableau la référence de la figure présentée dans la suite du rapport.

Tableau 5-2 Contextes d'accès à la localisation

Application	**Activité de l'utilisateur**	**Visibilité de l'application**	**État du mobile**	**Fonctionnalités sollicitées**	**Figure**
WhatsApp	Indifférent	Premier plan	Allumé	Localisation en direct et ponctuelle	5.1
Twitter	Indifférent	Premier plan	Allumé	Géolocaliser une publication	5.2
LinkedIn	Indifférent	Premier Plan	Allumé	Zip code	5.3
Skype	Marche	Premier Plan	Allumé	Localisation ponctuelle	5.4
Snapchat	Marche	Premier Plan	Allumé	Snap Map par intermittences	5.5 (G)
Snapchat	Arrêt	Premier Plan	Allumé	Snap Map en continu	5.5 (D)
Messenger	Arrêt	Premier plan	Allumé	Localisation en direct et ponctuelle	5.6
Facebook	Marche	Premier plan	Allumé	Amis à proximité et Géolocaliser une publication	5.7 (G)
Facebook	Marche	Premier plan	Allumé	Aucune en lien avec la localisation	5.7 (D)
Messenger	Marche	Premier plan	Allumé	Localisation en direct et ponctuelle	5.8 (G), 5.12 (G)

Application	Activité de l'utilisateur	Visibilité de l'application	État du mobile	Fonctionnalités sollicitées	Figure
Messenger	Arrêt	Premier plan	Allumé	Localisation en direct et ponctuelle	5.12 (D)
Messenger	Marche	Indifférent	En veille	Localisation en direct	5.9
Messenger	Marche	Indifférent	En veille	Localisation en direct	5.9
Instagram	Indifférent	Premier plan	Allumé	Géolocaliser une publication	5.10 (G)
Instagram	Indifférent	Arrière-plan	Allumé	Aucune	5.10 (D)
Instagram	Indifférent	Indifférent	En veille	Aucune	5.11

5.5.2 Résultats globaux

Tableau 5-3 Comportement de collecte des applications

	Messagerie texte	Localisation
Intrusives	Twitter	Instagram, Facebook, Messenger
Intrusives mais annoncées	Messenger	Snapchat
Respectueuses	Facebook, WhatsApp, Skype, Instagram, Snapchat	Twitter, LinkedIn, Skype
Exemplaire		WhatsApp

Le tableau 5.3 résume nos observations et résultats sur les comportements de collecte des applications testées. Les applications sont intrusives si elles collectent des données dans un contexte non justifié pour l'utilisateur. Par exemple, Facebook accède aux données de position, sans qu'aucune de ses fonctionnalités en lien avec la localisation ne soit sollicitée. Il est possible que l'application ait effectivement ce besoin fonctionnel pour personnaliser le contenue. Cependant, pour ne pas rentrer en conflit avec les attentes utilisateur, l'application devrait, comme le fait Snapchat, l'annoncer ce qui permettrait à l'utilisateur de revoir ses attentes à la baisse. Les applications sont dites respectueuses quand les accès effectifs sont une conséquence directe d'une action utilisateur, comme un tweet localisé. Enfin WhatsApp est dite exemplaire car elle optimise la collecte en minimisant le volume et la fréquence de collecte des données de localisation.

5.5.3 Accès à la messagerie texte

Les vérifications du numéro de mobile mises à part, seules deux applications accèdent aux informations concernant les messages texte : Messenger et Twitter.

Ces accès sont attendus dans le cas de Messenger, car elle est utilisée comme application de messagerie par défaut. Elle traite donc différents Intents pour être avertie de la réception d'un message (*« SMS_RECEIVED »*) recevoir les messages entrants (*« SMS_DELIVER »*, *« WAP_PUSH _DELIVER »*) et expédier les messages sortants (*« ACTION_SENDTO »*, *« ACTION_SEND »*). De plus, elle accède aux fournisseurs de contenu systèmes de la messagerie texte en lecture (*query*) et en écriture (*insert, update, delete*).

Nous avons pu constater que si l'application est en arrière-plan ou que le téléphone est en veille Messenger accède en lecture aux fournisseurs de contenus de SMS/MMS à une fréquence de sept fois par minute. Aussi, si l'utilisateur est en consultation active de ses messages, à savoir si l'application est en premier plan, la fréquence peut atteindre les 20 accès par minute. Les accès en écriture et gestion des Intents sont ponctuels et en adéquation avec les attentes utilisateur : conséquences directes d'une action de sa part (rédaction / modification / suppression de message) ou réception d'un nouveau message.

Concernant Twitter, les accès sont plus difficiles à interpréter : cette application fait une écoute active de la diffusion des SMS, « SMS_RECEIVED ». Or cette diffusion est destinée aux applications qui doivent lire des messages entrants spéciaux, tels que la vérification du numéro de téléphone. À défaut d'avoir pu le tester, nous supposons que l'écoute est liée à la fonctionnalité d'alertes par SMS.

5.5.4 Accès à localisation

Toutes les applications ont dans leur manifeste au moins une permission du groupe *Localisation*. Nous présentons ci-après différentes observations propres à chaque application.

Comme déjà dit, **Pinterest** n'a pas de fonctionnalité propre à la localisation, et de fait, en pratique, même en lui accordant l'autorisation *« ACCESS_FINE_LOCATION »*, l'application ne sollicite aucunes des méthodes et mécanismes d'Intents que nous traçons. Dès lors, soit Pinterest utilise une méthode peu ou pas documentée d'accès à la localisation (et donc non présente dans notre outil de traces), soit elle pratique la sur-réclamation de permissions.

L'application **WhatsApp** utilise deux méthodes d'accès à la localisation selon la fonctionnalité sollicitée (*Voir* figure 5.1). Si l'utilisateur fait un partage de localisation ponctuel, WhatsApp fait un appel ponctuel, par Intents, au service de localisation Google interne pour fournir une carte où l'utilisateur est pointé. Si l'utilisateur fait un partage de localisation en direct (temps réel) WhatsApp interroge les fournisseurs *GPS* et *Network* afin de déterminer la localisation du mobile. Ces accès ne sont pas pour autant continus pendant toute la durée de partage. WhatsApp n'accède, de facto, à la localisation que quand cela est nécessaire : l'un des participants de la conversation, dans laquelle le partage de localisation a lieu, utilise activement la fonctionnalité, à savoir regarde la carte. WhatsApp respecte donc, sans équivoque, l'intégrité contextuelle.

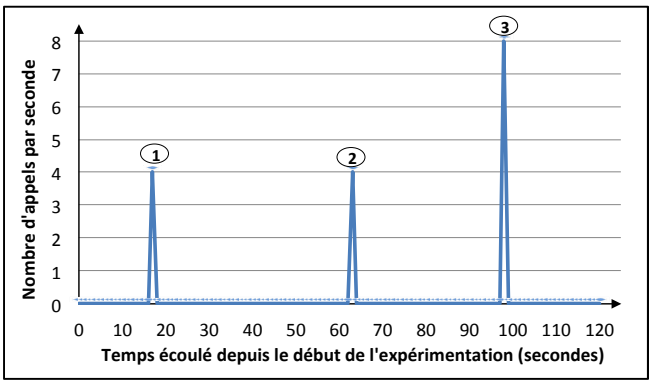

Figure 5.1 Accès par WhatsApp :
(1) Carte affichée sur le mobile de l'émetteur;
(2) Carte affichée sur le mobile d'un récepteur;
(3) Envoi ponctuel de la localisation

L'application **Twitter** est assez raisonnable au travers du prisme de l'intégrité contextuelle : si l'utilisateur géolocalise un tweet, elle sollicite ponctuellement, par Intents, le service de

localisation Google interne (*Voir* figure 5.2). De plus, elle accède exceptionnellement au fournisseur GPS (première connexion au service, allumage du mobile) afin de personnaliser le contenu en fonction de la localisation. Nos mesures n'ont pas permis de déterminer la fréquence ou les catalyseurs précis pour ce second type d'accès.

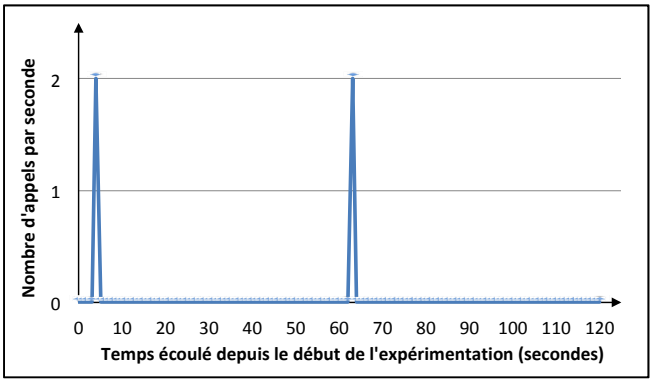

Figure 5.2 Géolocalisation d'un tweet sur Twitter

Pour l'application **LinkedIn,** l'accès à la localisation, via le fournisseur *Fused*, survient immédiatement après la sollicitation de la fonctionnalité permettant d'informer automatiquement le code ZIP dans les coordonnées du profil. Néanmoins, il est intéressant de noter que l'outil de trace a dénombré 18 appels en l'espace de 20 millisecondes. Comme déjà expliqué, cette répétition d'appel n'apporte pas d'information supplémentaire et ne constitue pas en soi un défaut d'intégrité contextuelle. De fait, il est impossible que la donnée varie en 20 millisecondes (*Voir* figure 5.3). La répétition pourrait néanmoins être symptomatique d'une étrange implémentation de l'application LinkedIn.

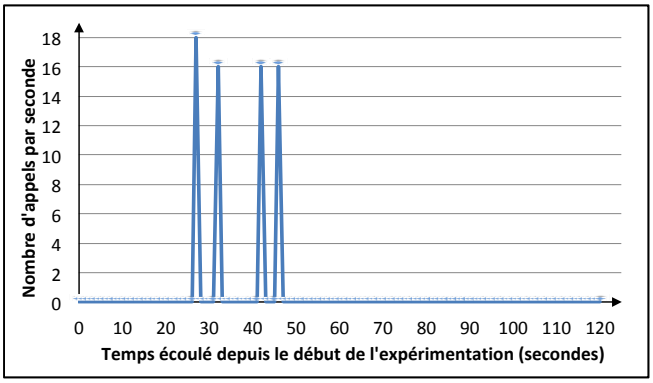

Figure 5.3 Accès multiples par LinkedIn

La fonctionnalité de partage de localisation ponctuelle de **Skype** sollicite une importante activité d'accès à la localisation (*Voir* figure 5.4) : appels au service de localisation Google par Intents, demandes de mise à jour des donnés des fournisseurs *GPS* et *Network* (*requestLocation Updates*, *getLastKnownLocation*) et écoute de notifications du gestionnaire de localisation (*LocationManager*) diffusées lorsque la position du mobile a changé (*onLocation Changed*). Néanmoins, comme pour le cas de LinkedIn, ces accès sont corrélés à l'utilisation de la fonctionnalité et donc en accord avec les attentes utilisateur. Aussi les tests ont révélé que si l'application est mise en arrière-plan alors que la fonctionnalité est en cours, l'accès est immédiatement arrêté. Il en va de même pour le téléphone en veille. On peut donc conclure qu'il n'y a pas ici de défaut d'intégrité contextuelle.

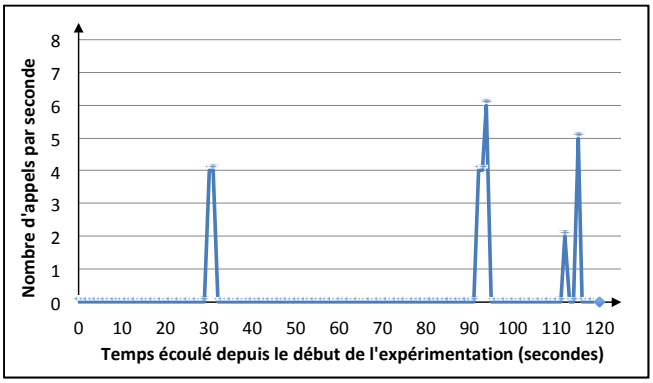

Figure 5.4 Accès par Skype

On observe un comportement similaire pour l'application **Snapchat** : si l'utilisateur utilise la fonctionnalité Snap Map, l'application demande des mises à jour aux fournisseurs *GPS* et *Network* et s'enquiert très fréquemment d'un possible changement de position (toutes les 10 secondes). Néanmoins, ces accès persistent tant que l'application reste en premier plan, et ce même avec la fonctionnalité fermée (*Voir* figures 5.5). En termes d'intégrité contextuelle, cela peut être problématique. En effet, lors de l'utilisation de la fonctionnalité Snap Map, une telle fréquence de collecte peut sembler justifiée, mais cela n'est plus forcément le cas lorsqu'il s'agit de prendre des photos. Cependant, il est important de noter que lorsque l'utilisateur concède les droits d'accès à la localisation, il est averti que cela est utilisé pour différentes fonctionnalités (filtres géolocalisés, recherches, Snap Map et autres) et ce, tant que l'application est ouverte.

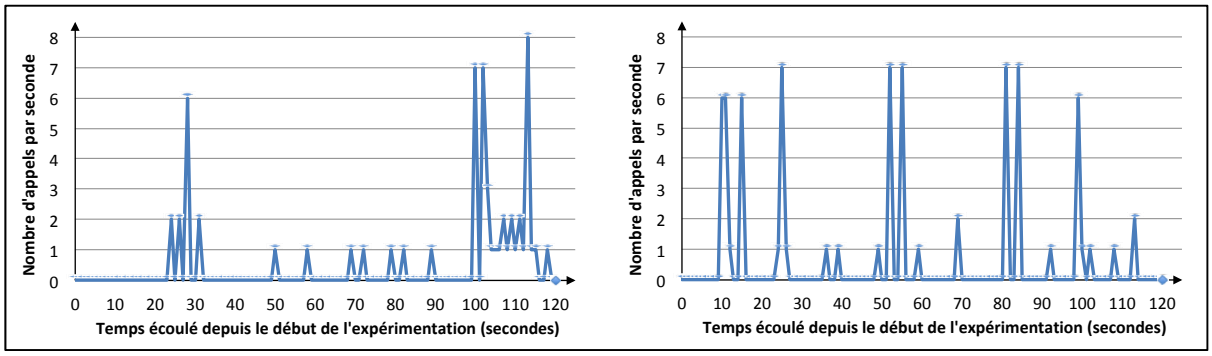

Figure 5.5 Accès à *Snap Map* : à gauche par intermittence, à droite, en continu

Les applications **Facebook** et **Messenger** ont un comportement d'accès à la localisation semblable. Elles collectent en continu la position du mobile (*Voir* figure 5.6) en utilisant le fournisseur *Network*, et plus précisément les données du réseau mobile (*getCellLocation*, *onCellLocation Changed*, *getNeighboringCellInfo*). Quand l'utilisateur emploie une fonctionnalité en lien direct avec la localisation, telle que *géomarquer une publication* sur Facebook ou *partager la localisation (ponctuelle ou en direct)* sur Messenger, les applications ont, en plus, recours au fournisseur GPS.

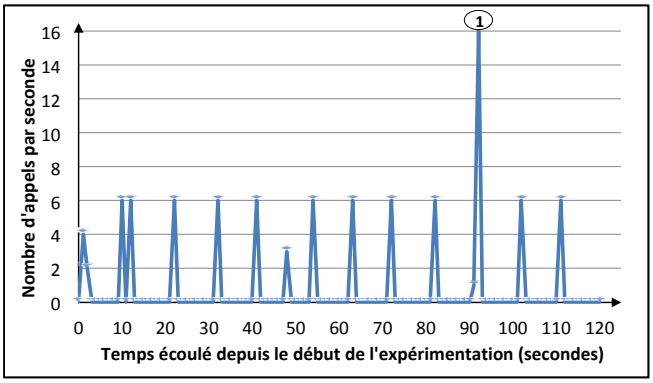

Figure 5.6 Accès par Messenger : sollicitation de la fonctionnalité partage de la localisation en direct (pics réguliers) et ponctuel (pic 1)

Nous avons fait des tests dans des contextes différents. Tout d'abord, avec les fonctionnalités nécessitant des données de localisation (Facebook : « Amis à proximité », Messenger : « Partage de localisation en direct »), puis sans. Nous avons constaté que bien que le second cas génère moins d'accès, il y a quand même des accès réguliers à la localisation, toutes les 30/40 secondes, et ce, sans besoin apparent (*Voir* figure 5.7). Facebook souhaite peut-être connaître en permanence la localisation de l'utilisateur pour adapter le contenu délivré. Ensuite, nous avons testé les fonctionnalités de localisation continues en faisant varier la visibilité de l'application et l'état du mobile. Nous avons constaté que ces changements d'état n'avaient que peu d'impact sur la fréquence de données collectées (*Voir* figure 5.8 et 5.9).

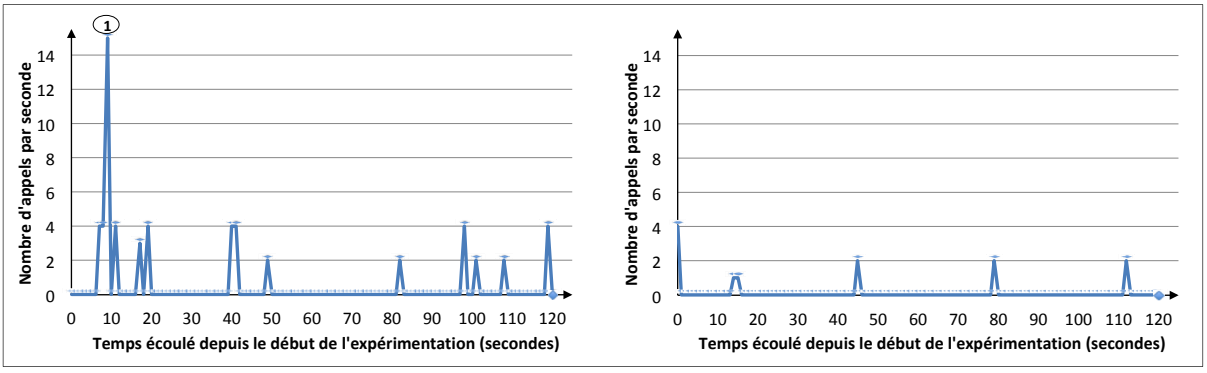

Figure 5.7 Accès par Facebook : à gauche, avec les fonctionnalités « géolocalisation d'une publication » (pic 1) et « amis à proximité » (petits pics); à droite, sans fonctionnalité

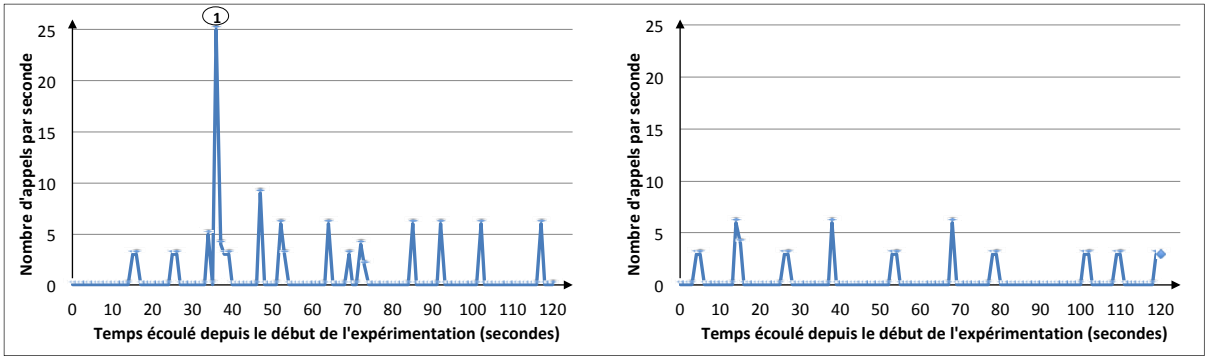

Figure 5.8 Accès par Messenger en faisant varier la visibilité : à gauche, premier plan; à droite arrière-plan avec fonctionnalités de localisation actives

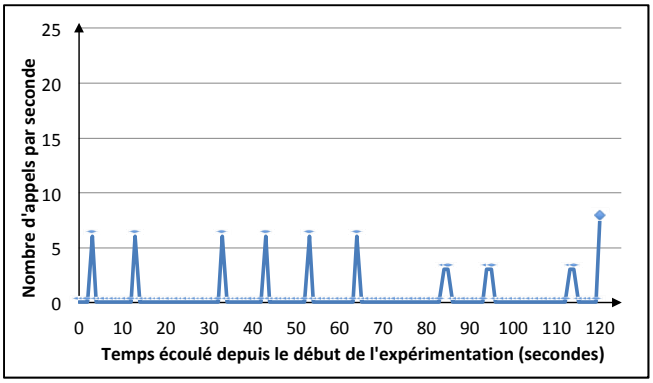

Figure 5.9 Accès par Messenger mobile en veille : avec fonctionnalités de localisation actives

Dès lors, nous pouvons affirmer que **Facebook** et **Messenger** présentent des **défauts d'intégrité contextuelle**.

Enfin, l'application **Instagram** est de loin l'**application la plus problématique**. Non seulement, comme déjà vu dans l'étude a priori, elle ne déclare pas clairement collecter la localisation, mais y accède en continu (à raison d'une fois par seconde), quel que soit sa visibilité (premier plan, arrière-plan) ou l'état du mobile (allumé, en veille) (*Voir* figures 5.10 et 5.11). De plus, elle sollicite chaque seconde systématiquement les trois fournisseurs (*GPS, Network* et *Passive*).

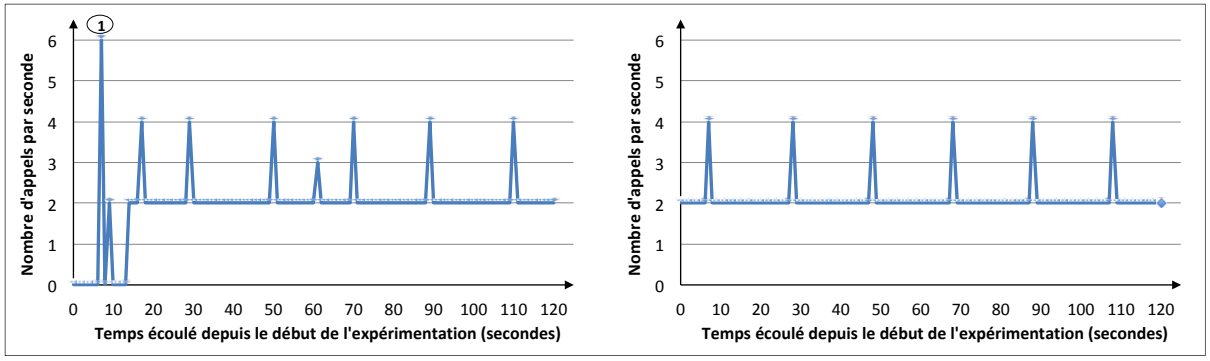

Figure 5.10 Accès par Instagram en faisant varier la visibilité : à gauche, premier plan avec géolocalisation d'une photographie (pic 1), à droite arrière-plan

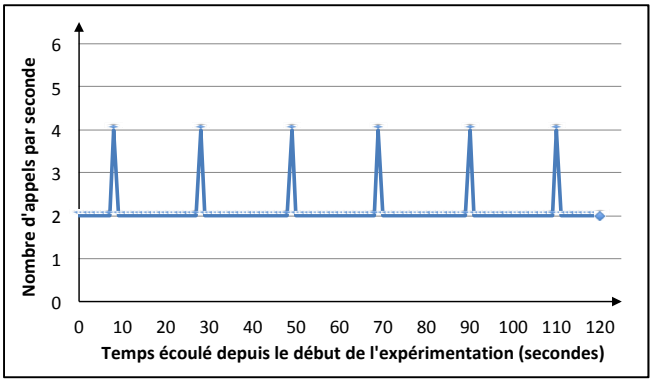

Figure 5.11 Accès par Instagram mobile en veille

Plus généralement, nous avons vérifié, que, conformément à la documentation Android (Android Developpers : Location Manager, 2018), certaines APIs adaptaient la fréquence de collecte à la vitesse de déplacement du mobile (*onLocationChanged, onCellLocation Changed*). La fréquence d'accès aux données de localisation peut donc être corrélée à l'activité utilisateur (statique, à pied, à vélo ou en transport motorisé). Ainsi, plus les déplacements sont rapides, plus les applications utilisant des méthodes de ces APIs (Skype, Snapchat, Messenger, Facebook et Instagram) ont une fréquence de collecte élevée (*Voir* figure 5.12).

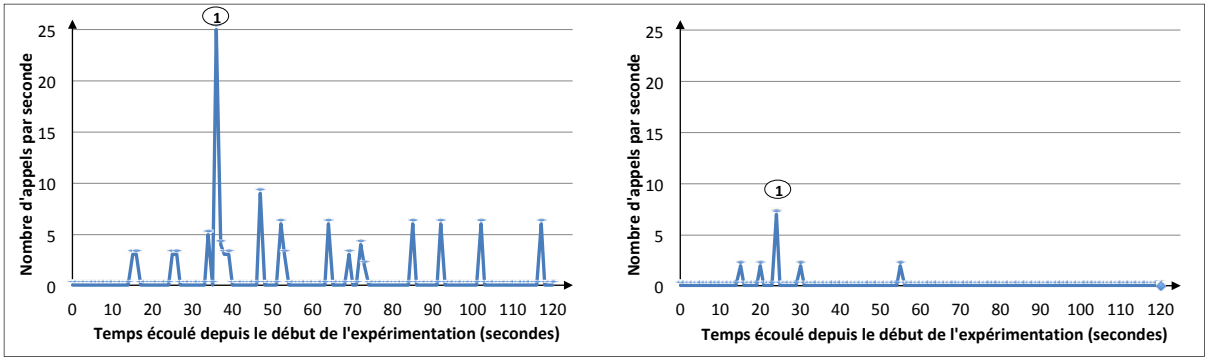

Figure 5.12 Accès par Messenger en faisant varier l'activité utilisateur : à gauche, en marche; à droite, à l'arrêt avec un partage ponctuel (pic 1) et continu (autres pics)

Aussi la méthode publique *requestLocation Updates (provider, minTime, minDistance, listener)* permet aux développeurs de préciser une fréquence de mise à jour en fonction du temps et/ou de la distance parcourue. Par exemple Instagram demande une mise à jour toutes les secondes, quelle que soit l'activité de l'utilisateur. Bien que certaines applications ou fonctionnalités justifient une telle précision de pistage, l'utilisateur devrait pouvoir être averti et avoir des moyens de réaction en cas de désaccord.

CHAPITRE 6

RESTRICTION DES ACCÈS SELON LE CONTEXTE

6.1 Améliorations viables du système d'autorisations

6.1.1 Rappels

Dans ce mémoire, il a été question de mettre en avant le problème de défaut d'intégrité contextuelle : trop souvent, les applications accèdent à des informations dans un contexte qui défie les attentes de l'utilisateur. Le système actuel d'autorisations est encore trop limité dans l'effort de contextualisation. Seules les autorisations dangereuses sont accordées dynamiquement lors du premier accès à la ressource protégée. En pratique, nous avons pu constater que ces demandes n'étaient pas toujours corrélées à un accès effectif à la ressource protégée, certaines applications demandent plusieurs groupes d'autorisations dès la première connexion au service. Lors d'une telle demande, l'utilisateur est convié à accorder un groupe d'autorisations dangereuses à une application donnée. Il est possible que l'utilisateur ne comprenne pas la demande, mais accepte pour avoir accès à la fonctionnalité, ou encore, qu'il trouve la demande justifiée, mais exprime surprise et confusion en constatant que l'application l'utilise dans des contextes différents. Les moyens d'agir en conséquence sont très limités : l'utilisateur peut révoquer les groupes d'autorisations dangereuses qui le dérangent dans les paramètres et les réactiver au besoin.

Ici, quand il est question de consentements via les groupes d'autorisations, nous adoptons le modèle conceptuel de l'utilisateur : le modèle mental qu'il a d'une demande d'autorisations permettant de prédire les effets d'un accord ou d'un refus et de comprendre les résultats. Ainsi, si une application demande l'accès au groupe *Localisation* pour permettre de géomarquer une photo, la demande est légitime. Cependant, cela cesse d'être le cas si, comme Instagram, l'application traque l'utilisateur et collecte ses données GPS tous les *n* mètres ou à intervalle de temps régulier même avec le téléphone en veille.

Le modèle conceptuel est basé sur différents facteurs : l'adéquation des permissions par rapport aux fonctionnalités, la fréquence d'accès aux ressources protégées, les préférences utilisateur en termes de protection de la vie privée, du contexte d'utilisation de l'application (fonctionnalité utilisée, activité utilisateur en cours), de la visibilité de l'application (premier plan, arrière-plan, fermée), etc. Il serait donc judicieux que les paramètres des permissions prennent en compte ces facteurs.

6.1.2 Obtenir un consentement éclairé

Nativement, les mobiles Android ne présentent pas de retour d'informations qui permet de surveiller le « comportement » des applications. L'utilisateur n'est pas informé des accès effectifs aux ressources ni du partage de ses informations personnelles entre applications. Nos résultats statistiques concordent avec ceux d'études antérieures (Almuhimedi et al., 2015; Wijesekera et al., 2015) : la collecte est trop intensive pour solliciter un aval de l'utilisateur au cas par cas de chaque accès ou l'informer tous les X accès (*Voir* figure 6.1). Cependant, présenter des statistiques de collecte à l'utilisateur permet de faire évoluer son modèle conceptuel et peut l'amener à questionner et revoir les autorisations concédées aux applications. Bien que ça ne soit pas l'objectif premier de ce mémoire, nous avons pris en considération les résultats des études se heurtant au modèle conceptuel des utilisateurs afin de proposer une solution viable. Nous pensons que les statistiques d'accès aux ressources protégées doivent être calculées en considérant le contexte afin d'aider à un paramétrage des autorisations plus proche des attentes utilisateur. Cela implique, bien sûr, qu'il y ait un paramétrage possible des autorisations selon le contexte.

6.1.3 Lier les autorisations au contexte

L'idée n'est pas nouvelle, le système d'autorisations iOS est sur ce point plus avancé que celui d'Android. Pour une application demandant l'accès à la localisation, le système donne trois options d'autorisations : jamais, lorsque l'application est active ou toujours.

Sur Android il est envisageable de faire quelque chose de similaire en prenant en considération la visibilité : si l'application est en premier plan, en arrière-plan ou fermée. Le consentement serait donc propre à chaque 3-uplet : groupe d'autorisations, application, visibilité. Comme une demande dynamique de consentement n'est possible que pour une application en premier plan, il faudrait un outil permettant de paramétrer les autorisations pour les autres cas : un gestionnaire d'autorisations natif. Comme pour le système actuel, les développeurs seraient libres de préciser lors de la demande dynamique pourquoi les fonctionnalités nécessitent des autorisations d'accès. De plus, le pourquoi de l'accès à une ressource protégée pourrait être précisé pour chaque 3-uplet (groupe, application visibilité) et reporté dans le gestionnaire d'autorisations.

Considérons la fonctionnalité « amis à proximité » de Facebook qui a la particularité de collecter la position du mobile en continu même si l'application n'est pas activement utilisée. Alors, dans notre solution le 3-uplet (*Localisation*, Facebook, arrière-plan) se verrait associer l'explication de son utilité pour la fonctionnalité « amis à proximité ».

On pourrait également imaginer que pour les cas de collecte continue, le gestionnaire d'autorisations proposerait un paramètre afin que l'utilisateur puisse choisir, selon son appréciation, entre différentes fréquences possibles d'accès aux ressources sensibles selon le contexte.

6.2 Test de faisabilité

Afin de savoir si une telle solution est possible, nous avons développé une application ***BlockPermBG*** qui détecte quand une application n'est plus en premier plan et lui retire automatiquement les permissions présélectionnées.

Cette application a deux objectifs : montrer que le cadriciel Android permet à ce jour de proposer une solution de gestionnaire d'autorisations plus transparent et proche des attentes utilisateur et étudier l'impact d'un blocage sur les fonctionnalités des applications étudiées.

6.2.1 Exigences techniques

La version 4.3 d'Android a introduit un gestionnaire d'autorisations à grains fins *AppOps*, masqué par défaut, qui a été retiré à la version 4.4.2. Accessible via une application externe, ce gestionnaire permettait à l'utilisateur d'accorder ou révoquer des autorisations individuellement pour les applications installées sur le mobile. Bien que caché et non accessible aux développeurs d'applications tierces, le gestionnaire est toujours maintenu et opérationnel dans le cadriciel Android pour les applications système.

Notre application ***BlockPermBG*** utilise des méthodes privées du cadriciel Android pour interagir avec *AppOps*. Pour ce faire, nous avons dû lui concéder les droits administrateur sur le mobile. Aussi, *AppOps* n'étant accessible qu'aux applications système, il a été nécessaire de rendre ***BlockPermBG*** système. Pour cela, nous avons utilisé le module *AppSystemizer* du cadriciel déjà cité *Magisk Manager* (*Voir* section 5.1).

Nous avons pu établir qu'en pratique l'application ***BlockPermBG*** est efficace : elle permet de retirer dynamiquement les permissions aux applications en arrière-plan et les rétablir quand ces dernières repassent en premier plan (*Voir* figures 6.1 et 6.2). En effet, on peut remarquer dans la figure 6.1 que les accès de certaines applications ne sont pas circonscrits à l'intervalle de temps où elles sont au premier plan (l'intervalle est indiqué par la légende du graphique en haut). C'est le cas, par exemple d'Instagram, mais aussi Facebook, Messenger, ce qui est cohérent avec les mesures exposées dans le chapitre 5. Avec BlockPermBG, on observe bien des accès restreints aux périodes où chaque application est au premier plan.

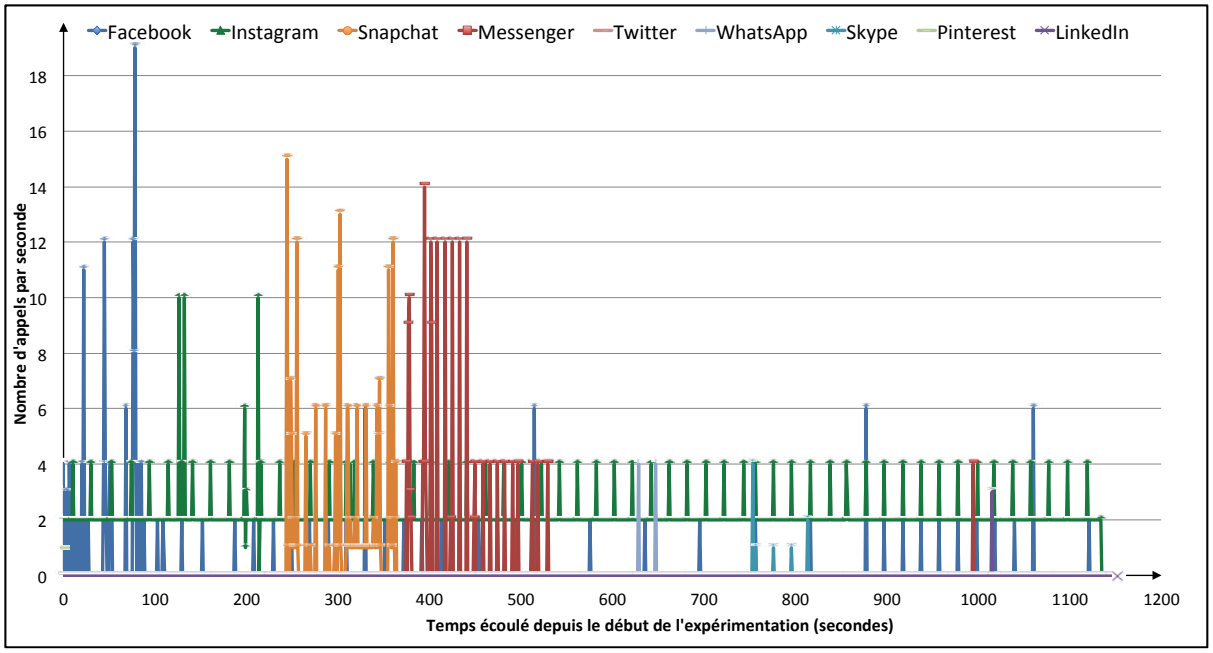

Figure 6.1 Sans BlockPermBG, accès à la localisation par les 9 applications, successivement en premier plan (120 secondes/application) dans l'ordre indiqué dans la légende du graphique

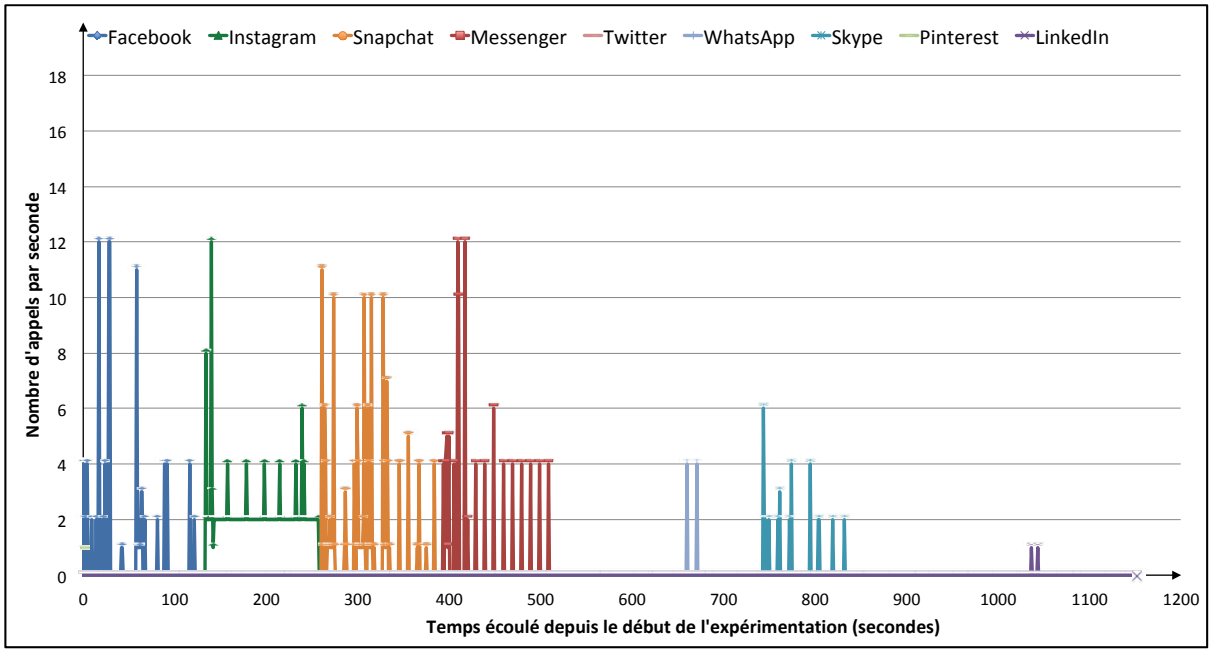

Figure 6.2 Avec BlockPermBG, accès à la localisation par les 9 applications, successivement en premier plan (120 secondes/application) dans l'ordre indiqué dans la légende du graphique

Cependant, certaines méthodes générant des rappels (*callbacks*) peuvent passer outre notre blocage. Quand ces méthodes sont appelées, un mécanisme de rappel est mis en place et lors de la révocation des permissions ce mécanisme n'est pas détruit. En pratique, la documentation Android explique que ces mécanismes sont automatiquement détruits lors l'arrêt du processus récepteur. Nous estimons donc que pour pallier ce problème il serait nécessaire, comme pour la procédure actuelle de révocation d'autorisations via les paramètres, de redémarrer le processus dédié de l'application.

Ce mécanisme d'arrêt est une façon pour Android d'avoir la garantie que l'application ne pourra pas continuer à recevoir des données de ressources protégées après révocation de ses droits d'accès (retrait de permissions). Il est actuellement adapté au système, car il est peu fréquent qu'un utilisateur révoque les droits d'une application en activité. Cependant, dans le cas où Android adopte une solution de gestionnaire d'autorisations tel que proposé dans ce mémoire, il serait prudent de mettre en place un autre mécanisme. En effet, lorsque le processus est redémarré, l'application est réinitialisée : dès lors, l'utilisateur perd le fil de sa navigation. Comme le gestionnaire lie les autorisations au contexte, cette réinitialisation serait quasi systématique et donc pénible pour l'utilisateur.

6.2.2 Évaluation de l'impact sur les fonctionnalités des applications

6.2.2.1 La messagerie texte

L'utilisation de notre solution de blocage a un impact non négligeable sur l'application Facebook Messenger, quand cette dernière est utilisée en tant qu'application de messagerie standard (SMS/MMS) par défaut. En effet, si l'application n'est pas en premier plan lors de la réception d'un message, elle ne peut pas lire les messages entrants : notre application ***BlockPermBG*** lui a révoqué les permissions requises. Ce défaut peut être facilement corrigé en utilisant dans le gestionnaire d'autorisations, comme pour le système actuel, une boîte de dialogue informant l'utilisateur qu'une telle révocation peut impacter sur la fonctionnalité de base de l'appareil : lire les SMS entrants.

La fonctionnalité «alertes par messages» de Twitter n'a pas pu être testée : aucun évènement marquant n'ayant eu lieu pendant la période d'expérimentation. Néanmoins, nous estimons que la fonctionnalité ne serait pas impactée, car il s'agit de recevoir des messages. Les autorisations impliquées sont donc propres à l'application de messagerie par défaut et non celles de Twitter.

Notre solution n'impacte pas la fonctionnalité de vérification automatique du numéro de mobile, car l'application reste au premier plan et a donc les permissions d'accès requises. De plus, si l'utilisateur révoque le groupe de permissions *Messagerie texte* quel que soit le contexte il devra, de lui-même, copier-coller le code de confirmation reçu par message texte, mais le gain de confidentialité en vaut probablement la peine.

6.2.2.2 La localisation

Au niveau de la localisation, notre solution a un impact minime sur les fonctionnalités des applications. Le blocage concerne uniquement trois des applications étudiées : Instagram, Messenger et Facebook qui accèdent à la localisation en arrière-plan.

Pour Instagram, cela n'a aucun impact et permet de réduire la consommation de la batterie. En effet, lorsque l'application repasse en premier plan, elle peut de nouveau accéder à la localisation et donc proposer la fonctionnalité de géotag et du contenu personnalisé.

Pour Messenger, le blocage en arrière-plan suspend le partage de localisation en direct. Cela peut être gênant dans le sens où il y a au moins deux utilisateurs impliqués dans l'échange : l'émetteur, source des données de localisation, et le récepteur. Si l'émetteur met Messenger en arrière-plan, le récepteur ne reçoit plus de mise à jour de la position. Il en résulte que la position affichée reste statique et est donc potentiellement erronée. Une solution viable serait d'utiliser un procédé similaire à celui de WhatsApp : informer l'ensemble des participants, prenant part à l'échange, de l'heure de la dernière mise à jour et conditionner les accès à la

localisation du mobile « pisté » au besoin, à savoir quand l'expéditeur ou le récepteur utilisent, de facto, la fonctionnalité.

Pour Facebook, l'incidence d'une révocation des permissions en arrière-plan est plus difficile à apprécier, car l'utilité des fonctionnalités impactées est très subjective. Aussi, Facebook est vaste et a de nombreuses fonctionnalités qui n'ont pas pu toutes être testées. La fonctionnalité « Amis à proximité » perd de son intérêt si elle est conditionnée au fait que l'ami à proximité doit, de surplus, être en train d'utiliser l'application en premier plan. Cependant une solution plus aboutie, tel qu'un gestionnaire natif proposé permettrait, comme expliqué en section 6.1.3, d'instruire l'utilisateur en lui fournissant les explications nécessaires pour qu'il soit maître de ses données et éclairé dans ses choix de partage de localisation ou toute autre donnée sensible.

Les résultats sont concluants, la grande majorité des accès sont effectivement bloqués et la perte fonctionnelle est minime et facilement compensable. De plus la charge décisionnelle supplémentaire est dérisoire : pour chaque couple (application, groupe d'autorisation) l'utilisateur configure blocage ou non des accès en arrière-plan. Par exemple, pour Messenger, utilisée comme application de messagerie par défaut, il conviendrait de ne pas bloquer le groupe de *Messagerie texte*, pour pouvoir recevoir des messages, quelle que soit la visibilité de l'application. Le groupe d'autorisations *Localisation* pourrait cependant lui être retiré en arrière-plan sans aucun dommage.

CONCLUSION

La recherche présentée dans ce mémoire est centrée sur les défauts d'intégrité contextuelle, définie comme l'écart entre les attentes d'un utilisateur et l'accès effectif aux ressources du mobile et à ses données personnelles, appliquée à dix applications hégémoniques de médias sociaux sur Android : Facebook, YouTube, Messenger, Instagram, Snapchat, WhatsApp, Skype, Pinterest, Twitter et LinkedIn. L'objectif était d'étudier dans sa globalité le modèle actuel de consentement utilisateur d'Android au travers du prisme des attentes utilisateur concernant la collecte des données. Pour cela nous avons procédé en trois étapes : une étude a priori pour cerner les pratiques de collecte annoncées, une étude empirique pour examiner le comportement réel de collecte de données des applications et identifier les conditions susceptibles de défier les attentes utilisateur, et enfin une étude de la viabilité d'une solution permettant de palier à ce défaut d'intégrité contextuelle.

L'étude a priori est centrée sur les indicateurs des pratiques de collecte d'informations disponibles sur la plateforme Android et son marché officiel Google Play. L'objectif était d'identifier les obstacles à la compréhension de l'utilisateur concernant la collecte de données et les défauts de légitimité du consentement subséquent. La méthodologie suivie peut se résumer en collecte, exploration et interprétation des données utiles. Ces données étaient de trois types : les sections relatives à la collecte de données des politiques de confidentialité des dix applications de médias sociaux, les listes des autorisations requises présentées sur Google Play, et les listes de permissions extraites des dix manifestes.

Nous avons bâti une pseudo-ontologie afin de décrire et comparer les politiques entre elles sur deux caractéristiques principales : la clarté et la complétude d'énonciation des types de données collectées et la présence d'informations concernant les choix et contrôle possibles sur cette collecte. Cela a permis de mettre en évidence les défauts, non négligeables, de ce mode d'information à l'utilisateur : les politiques sont longues, vagues, inclusives, difficiles à lire et à comprendre. L'information utile est noyée dans un langage entre juridique et technique peu accessible à un utilisateur lambda.

L'analyse détaillée des permissions / autorisations des dix applications a mis en exergue différents défauts d'information à l'utilisateur. Premièrement, pour une application donnée, il y a systématiquement plus de permissions que d'autorisations. En réalité, cela est bien souvent lié au modèle et version du cadriciel Android de l'appareil. Plus problématique, l'écart entre le nombre de permissions et autorisations est parfois lié au fait que certains développeurs déclarant des permissions spécifiques ne se donnent pas la peine de les traduire en autorisations. Cela constitue une porte ouverte à des abus tels que le partage de ressources privées entre applications à l'insu de l'utilisateur. Enfin dans le cadriciel Android, certaines permissions très sensibles, dites signatures, n'ont pas d'équivalences en autorisations : elles sont parfois traduites en paramètres système, parfois en autorisations normales ou, plus grave, elles ne sont pas présentées du tout à l'utilisateur.

Le modèle peine à sensibiliser l'utilisateur aux risques encourus concernant la collecte de données et à l'aider à prendre des décisions avisées en termes de vie privée. En effet, les autorisations normales sont attribuées d'office et consultables pour des raisons de transparence. Aussi l'utilisateur est, à ce jour, uniquement sollicité pour concéder des autorisations dangereuses par groupe aux applications installées, et ce sans distinction de cas d'usage. Il peut également révoquer ces mêmes groupes, cependant, le cadriciel Android ne fournit pas d'outil d'évaluation du comportement des applications justifiant une telle révocation. L'utilisateur n'a donc pas les moyens d'être en contrôle de ses données personnelles.

L'étude pratique s'est concentrée sur les accès effectifs aux ressources protégées par les dix applications. L'objectif était d'évaluer l'adéquation entre le modèle conceptuel de l'utilisateur et la réalité concernant la collecte de données. En d'autres termes, il était de question de voir si le comportement des applications, accès effectifs aux ressources, défie les attentes utilisateur. Pour des raisons de faisabilité, nous avons restreint l'étude aux données de localisation et messagerie texte. La méthodologie adoptée s'est articulée en trois phases : développements d'outils de traces permettant d'enregistrer et quantifier les accès effectifs aux ressources, tests et mesures, extraction des mesures pertinentes et analyse. Des tests

préliminaires ont révélé l'importance d'un environnement contrôlé et un traitement automatisé des traces. Nous avons donc développé un outil permettant de capter les conditions expérimentales et des scripts Python et requêtes SQL automatisant la récupération et le traitement des traces générées dans une seule et même base de données. D'autre part, nous avons identifié des données contextuelles influant sur les accès effectifs et avons réalisé une batterie de tests en faisant varier ces paramètres expérimentaux : activité utilisateur, visibilité de l'application, état du mobile et fonctionnalités à stimuler.

Nous avons pu observer qu'à l'exception de Twitter qui fait une écoute active des messages systèmes concernant les SMS entrants, les accès aux données de la messagerie texte sont prévisibles, et donc en accord avec les attentes utilisateur. Concernant les données de localisation, les résultats sont plus hétérogènes. L'application WhatsApp n'accède à la localisation que quand c'est fonctionnellement nécessaire, à savoir quand ses fonctionnalités de partage de localisation sont activement sollicitées par l'utilisateur. Il en est de même pour l'application LinkedIn, Twitter et Skype. Moins rigoureuse, l'application Snapchat accède à la localisation uniquement si elle est ouverte et activement utilisée, à savoir au premier plan. Cela est néanmoins clairement informé à l'utilisateur qui pourra donc revoir ses attentes. Plus problématique, nous avons constaté que les applications Facebook et Messenger accèdent sporadiquement aux données de localisation sans raison apparente, leurs fonctionnalités liées à la localisation n'étant pas sollicitées lors des mesures. Aussi, lorsque ces fonctionnalités sont sollicitées les accès, bien que justifiables, peuvent troubler l'utilisateur. Ici le terme « justifiable » est employé pour signifier que le modèle est logique : si l'utilisateur veut recevoir des notifications quand un ami est à proximité, il est nécessaire de collecter sa localisation régulièrement. En effet, les applications continuent de collecter les données de localisation même si elles sont en arrière-plan ou que le mobile est en veille. Néanmoins, l'utilisateur n'est pas forcément au fait des besoins techniques et doit donc être prévenu. Plus abusif encore, l'application Instagram collecte les données de localisation du mobile toutes les secondes et ce quel que soit le contexte d'utilisation. Ici aucun besoin technique ne justifie de pister ainsi l'utilisateur : il y a clairement une atteinte à l'intégrité contextuelle.

Notre étude pratique a permis de révéler le besoin de retour d'informations qui permettrait de surveiller le comportement de collecte des applications. Avec un tel retour, l'utilisateur serait en mesure de perfectionner son modèle conceptuel, revoir ses attentes en termes de vie privée et paramétrer les autorisations concédées aux applications en connaissance de cause. Aussi, l'analyse a permis de déceler deux défauts structurels du cadriciel Android : le consentement dissimulé du mécanisme d'Intent substituant le contrôle d'accès du système par une action utilisateur, et le fournisseur *Passive* de données de localisation qui permet une collecte opportuniste sans réel besoin fonctionnel.

Finalement, la dernière partie de cette recherche a été dévouée à la proposition d'une preuve de concept. L'objectif était de montrer qu'il est possible de réduire les défauts d'intégrité contextuelle sans pénaliser lourdement les applications et avec un minimum de demandes à l'utilisateur. En d'autres termes, il est possible de concevoir une solution viable afin que les accès effectifs aux ressources ne soient pas en conflit avec le modèle conceptuel de l'utilisateur. La méthodologie adoptée s'est articulée en trois phases : conception, implémentation, et évaluation d'une solution permettant de paramétrer les permissions concédées aux applications en fonction du contexte d'utilisation.

Le modèle conceptuel de l'utilisateur repose sur différents facteurs : l'adéquation entre les permissions demandées et les fonctionnalités, la fréquence d'accès aux ressources protégées, la visibilité de l'application lors de l'accès, etc. Notre solution intègre l'un de ces facteurs, la visibilité de l'application lors de l'accès, au gestionnaire d'autorisations permettant à l'utilisateur de révoquer un groupe d'autorisations dangereuses à une application donnée en arrière-plan. Implémentée et testée sur les dix applications de médias sociaux et les groupes d'autorisations *Localisation* et *Messagerie texte*, la solution est viable et prometteuse : seulement quelques mécanismes de rappels passent outre le blocage et la perte fonctionnelle est minime. Dans les deux cas, cela est facilement compensable.

Les trois études de ce mémoire mettent en perspectives les limitations du système actuel de permissions lors de la collecte des données personnelles sur Android. Le consentement

utilisateur est trop souvent mal avisé due à trop peu d'informations accessibles : politiques difficiles à lire et comprendre, liste exhaustive des autorisations trop techniques, retour sur le comportement des applications quasi insignifiant. De plus, la plateforme ne fournit qu'une piètre latitude de contrôle natif : révocation manuelle des groupes d'autorisations dangereuses.

Il ne faut cependant pas oublier de souligner les efforts de transparence de plus en plus présents dans les plateformes mobiles de nos jours, tels que les indicateurs visuels qui permettent d'informer l'utilisateur en temps réel du fait que sa localisation est en train d'être collectée. Il est également possible de voir dans les paramètres quelles applications ont fait une demande récente de la localisation. Cependant, ces indicateurs manquent d'informations cruciales pour assister l'utilisateur dans le paramétrage des autorisations concédées aux applications : qui accède, pourquoi, à quelle fréquence, dans quel contexte, etc.

De nombreuses études dans le domaine incitent la plateforme à proposer des outils d'information et de contrôle plus adéquats afin que l'utilisateur appréhende mieux la collecte des données opérée insidieusement ou non par les applications installées. Nos mécanismes de traces et métriques ainsi que notre application de blocage des permissions en fonction de la visibilité de l'application pourraient constituer les prémisses d'un gestionnaire de confidentialité Android. Ce dernier, à l'instar des applications natives de contrôle de batterie actuelles, permettrait un contrôle de la collecte des données avec des indicateurs de type fréquence d'accès aux ressources, volume de données collectées, contexte d'accès, etc., et des paramétrages liés au contexte. Nous avons étudié le facteur visibilité de l'application, mais il est possible d'envisager d'autres paramètres contextuels tels que l'heure de la journée, le lieu, ou encore l'activité utilisateur.

Notre recherche pourrait servir de pilote pour des recherches plus approfondies, permettant de réaliser des études utilisateurs à grande échelle pour affiner le modèle conceptuel à considérer dans la spécification de gestionnaires natifs améliorés. Nous souhaitons également souligner que la démarche serait grandement allégée si elle venait des concepteurs et

développeurs du cadriciel Android. Ayant un accès interne aux fonctionnalités de base d'Android, ils ne seraient pas confrontés aux mêmes écueils et difficultés techniques : imputer l'accès à une ressource à une application donnée ou acquérir les droits systèmes doit être alors relativement simple. Aussi, cela éviterait que la solution ne soit accessible et utilisable que par une petite communauté d'utilisateurs capables de débrider leur mobile.

ANNEXE I

PERMISSIONS ET AUTORISATIONS INSTAGRAM

Tableau-A I-1 Catégorisation des permissions et autorisations d'Instagram (Version 39.0.0.36)

		Permissions (dangereuses en gras)	Autorisation (si affichée)
Standard		**android.permission.GET_ACCOUNTS**	**Rechercher des comptes sur l'appareil**
		android.permission.READ_CONTACTS	**Lire vos contacts**
		android.permission.ACCESS_FINE_LOCATION	**Accéder à votre position précise (GPS et réseau)**
		android.permission.RECEIVE_SMS	**Recevoir des messages texte (SMS)**
		android.permission.CALL_PHONE	**Appeler directement des numéros de téléphone**
		android.permission.READ_PHONE_STATE	**Voir l'état et l'identité du téléphone**
			Lire le contenu de la carte SD
		android.permission.WRITE_EXTERNAL_STORAGE	**Modifier ou supprimer le contenu de la carte SD**
		android.permission.CAMERA	**Prendre des photos et filmer des vidéos**
		android.permission.RECORD_AUDIO	**Enregistrer des fichiers audio**
		android.permission.ACCESS_NETWORK_STATE	Afficher les connexions réseau
		android.permission.BLUETOOTH	S'associer à des appareils Bluetooth
		android.permission.BLUETOOTH_ADMIN	Accéder aux paramètres Bluetooth
		android.permission.CHANGE_WIFI_STATE	Activer/désactiver la connexion Wi-Fi
		android.permission.INTERNET	Bénéficier d'un accès complet au réseau
		android.permission.MODIFY_AUDIO_SETTINGS	Modifer vos paramètres audio
		android.permission.READ_PROFILE	
		android.permission.RECEIVE_BOOT_COMPLETED	S'exécuter au démarrage
		android.permission.USE_CREDENTIALS	
		android.permission.VIBRATE	Gérer le vibreur
		android.permission.WAKE_LOCK	Empêcher le téléphone de passer en mode veille
		com.android.launcher.permission.INSTALL_SHORTCUT	Installer des raccourcis
		com.android.launcher.permission.UNINSTALL_SHORTCUT	Désinstaller des raccourcis
Non Standard	applicative	com.google.android.c2dm.permission.RECEIVE	Recevoir des données d'Internet
		com.instagram.direct.permission.DIRECT_APP_THREAD_STORE_SERVICE	
		com.instagram.direct.permission.PROTECTED_DEEPLINKING	
	constructeur	android.permission.UPDATE_APP_BADGE	
		com.amazon.device.messaging.permission.RECEIVE	
		com.htc.launcher.permission.READ_SETTINGS	
		com.htc.launcher.permission.UPDATE_SHORTCUT	
		com.huawei.android.launcher.permission.CHANGE_BADGE	
		com.sonyericsson.home.permission.BROADCAST_BADGE	
	propre	.permission.C2D_MESSAGE	
		.permission.RECEIVE_ADM_MESSAGE	

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Agarwal, Y., & Hall, M. (2013). ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. Dans *Proceeding of the 11th annual international conference on Mobile systems, applications, and services* (pp. 97-110). ACM.

Almuhimedi, H., Schaub, F., Sadeh, N., Adjerid, I., Acquisti, A., Gluck, J., Cranor, J.L., Agarwal, Y. (2015). Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. Dans *Proceedings of the 33rd annual ACM conference on human factors in computing systems* (pp. 787-796). ACM.

Andriotis, P., Sasse, M. A., & Stringhini, G. (2016). Permissions snapshots: Assessing users' adaptation to the Android runtime permission model. Dans *Information Forensics and Security (WIFS), 2016 IEEE International Workshop on* (pp. 1-6). IEEE.

Android Developpers (2017). System Permissions.
Repéré à http://developer.android.com/guide/topics/security/permissions.html

Android Developpers (2018). Intent Common.
Repéré à https://developer.android.com/guide/components/intents-common

Android Developpers (2018). Location Manager.
Repéré à https://developer.android.com/reference/android/location/LocationManager

Android Developpers (2018). Permissions Overview.
Repéré à https://developer.android.com/training/permissions

Android Developpers (2018). Write and View Logs with Logcat.
Repéré à https://developer.android.com/studio/debug/am-logcat

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., McDaniel, P. (2014). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices, 49*(6), 259-269.

Balebako, R., Jung, J., Lu, W., Cranor, L. F., & Nguyen, C. (2013). Little brothers watching you: Raising awareness of data leaks on smartphones. Dans *Proceedings of the Ninth Symposium on Usable Privacy and Security* (pp. 12). ACM.

Beresford, A. R., Rice, A., Skehin, N., & Sohan, R. (2011). Mockdroid: trading privacy for application functionality on smartphones. Dans *Proceedings of the 12th workshop on mobile computing systems and applications* (pp. 49-54). ACM.

Bilogrevic, I., Huguenin, K., Agir, B., Jadliwala, M., Gazaki, M., & Hubaux, J.-P. (2016). A machine-learning based approach to privacy-aware information-sharing in mobile social networks. *Pervasive and Mobile Computing, 25*, 125-142.

Bokhorst, M. (2012). XPrivacy (Version 3.6.19) [Logiciel]
Repéré à https://github.com/M66B/XPrivacy/blob/master/README.md

Bugiel, S., Heuser, S., & Sadeghi, A.-R. (2013). Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. Dans *USENIX Security Symposium* (pp. 131-146).

Cate, F. H. (2006). The failure of fair information practice principles.

Chia, P. H., Yamamoto, Y., & Asokan, N. (2012). Is this app safe?: a large scale study on application permissions and risk signals. Dans *Proceedings of the 21st international conference on World Wide Web* (pp. 311-320). ACM.

Chitkara, S., Gothoskar, N., Harish, S., Hong, J. I., & Agarwal, Y. (2017). Does this App Really Need My Location?: Context-Aware Privacy Management for Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 1*(3), 42.

Coen, R., King, J., & Wong, R. Y. (2016). The Privacy Policy Paradox. Dans *WPI@ SOUPS*.

Contextual Integrity. (2017). Dans *Wikipédia.*
Repéré le 15 septembre 2017 à https://en.wikipedia.org/wiki/Contextual_Integrity

Continella, A., Fratantonio, Y., Lindorfer, M., Puccetti, A., Zand, A., Kruegel, C., & Vigna, G. (2017). Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. Dans *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)* (pp. 1-16).

Crowdsourcing. (2018). Dans *Le Grand Dictionnaire de Terminologie*.
Repéré à http://gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=45436

Egelman, S., Felt, A. P., & Wagner, D. (2013). Choice architecture and smartphone privacy: There'sa price for that. Dans *The economics of information security and privacy* (pp. 211-236). Springer.

Elenkov, N. (2014). *Android security internals: An in-depth guide to Android's security architecture*. No Starch Press.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., Sheth, A. N. (2014). TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS), 32*(2), 5.

Facebook (2018). Facebook (Version 165.0.0.53.93) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.facebook.katana

Facebook (2018). Messenger (Version 158.0.0.2.96) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.facebook.orca

Facebook Inc. (2016). Data Policy. Repéré à https://m.facebook.com/about/privacy/

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. Dans *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 627-638). ACM.

Felt, A. P., Egelman, S., & Wagner, D. (2012). I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. Dans *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices* (pp. 33-44). ACM.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D. (2012). Android permissions: User attention, comprehension, and behavior. Dans *Proceedings of the eighth symposium on usable privacy and security* (pp. 3). ACM.

Frank, M., Dong, B., Felt, A. P., & Song, D. (2012). Mining permission request patterns from android and facebook applications. Dans *Data Mining (ICDM), 2012 IEEE 12th International Conference on* (pp. 870-875). IEEE.

Fuchs, A. P., Chaudhuri, A., & Foster, J. S. (2009). *Scandroid: Automated security certification of android*.

Google LLC (2017). Google Play Help : Control your app permissions on Android 6.0 and up. Repéré à https://support.google.com/googleplay/answer/6270602

Google LLC (2017). Privacy Policy. Repéré à https://www.google.com/policies/privacy/

Google LLC (2018). YouTube (Version 13.10.59) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.google.android.youtube

Google Play (2017) Centre d'informations règlementaires pour les développeurs : Confidentialité, sécurité et fraude.
Repéré à https://play.google.com/intl/fr/about/privacy-security-deception/index.html

Gordon, M. I., Kim, D., Perkins, J. H., Gilham, L., Nguyen, N., & Rinard, M. C. (2015). Information Flow Analysis of Android Applications in DroidSafe. Dans *NDSS*. Citeseer.

Gorla, A., Tavecchia, I., Gross, F., & Zeller, A. (2014). Checking app behavior against app descriptions. Dans *Proceedings of the 36th International Conference on Software Engineering* (pp. 1025-1035). ACM.

Haris, M., Haddadi, H., & Hui, P. (2014). Privacy leakage in mobile computing: Tools, methods, and characteristics. *arXiv preprint arXiv:1410.4978*.

Hornyack, P., Han, S., Jung, J., Schechter, S., & Wetherall, D. (2011). These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. Dans *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 639-652). ACM.

Instagram (2018). Instagram (Version 39.0.0.0.36) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.instagram.android

Instagram Inc. (s.d.). Privacy Policy. Repéré à https://help.instagram.com/155833707900388

Ismail, Q., Ahmed, T., Kapadia, A., & Reiter, M. K. (2015). Crowdsourced exploration of security configurations. Dans *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 467-476). ACM.

Jing, Y., Ahn, G.-J., Zhao, Z., & Hu, H. (2014). Riskmon: Continuous and automated risk assessment of mobile applications. Dans *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy* (pp. 99-110). ACM.

Kayes, I., & Iamnitchi, A. (2015). A survey on privacy and security in online social networks. *arXiv preprint arXiv:1504.03342*.

Krubhala, P., Niranjana, P., & Sindhu Priya, G. (2015). Online Social Network-a Threat to Privacy and Security of Human Society. *International Journal of Scientific and Research Publications, 5*(4).

Lawson, N. (2015). CatLog (Version 12.0.2) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.nolanlawson.logcat&hl

Lin, J., Amini, S., Hong, J. I., Sadeh, N., Lindqvist, J., & Zhang, J. (2012). Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. Dans *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (pp. 501-510). ACM.

Lin, J., Liu, B., Sadeh, N., & Hong, J. I. (2014). Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings.

LinkedIn (2018). LinkedIn (Version 4.1.142) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.linkedin.android

LinkedIn Corporation (2017). Privacy Policy.
Repéré à https://www.linkedin.com/legal/privacy-policy

Liu, B., Andersen, M. S., Schaub, F., Almuhimedi, H., Zhang, S. A., Sadeh, N., Aquisti, A., Agarwal, Y. (2016). Follow my recommendations: A personalized privacy assistant for mobile app permissions. Dans *Symposium on Usable Privacy and Security*.

Liu, F., Fella, N. L., & Liao, K. (2016). Modeling language vagueness in privacy policies using deep neural networks. Dans *2016 AAAI Fall Symposium Series*.

Liu, F., Wilson, S., Schaub, F., & Sadeh, N. (2016). Analyzing vocabulary intersections of expert annotations and topic models for data practices in privacy policies. Dans *2016 AAAI Fall Symposium Series*.

Maxfield, J. (2016). AppSystemizer (Version 1.6.0) [Logiciel] Repéré à https://forum.xda-developers.com/apps/magisk/module-app-systemizer-t3477512

McDonald, A. M., & Cranor, L. F. (2008). The cost of reading privacy policies. *ISJLP, 4*, 543.

Média Social. (2018). Dans *Le Grand Dictionnaire de Terminologie*.
Repéré à http://www.granddictionnaire.com/ficheOqlf.aspx?Id_Fiche=26502881

Microsoft Corporation (2018). Microsoft Privacy Statement.
Repéré à https://privacy.microsoft.com/en-ca/privacystatement

Murphy, M. L. (2017). Source code to omnibus edition of *_The Busy Coder's Guide to Android Development* [GitHub]
Repéré à https://github.com/commonsguy/cw-omnibus

Murphy, M. L. (2017). *The Busy Coder's Guide to Android Development 8.9*. United States: CommonsWare, 2017.

Nissenbaum, H. (2004). Privacy as contextual integrity. *Wash. L. Rev., 79*, 119.

Obar, J. A., & Oeldorf-Hirsch, A. (2016). The biggest lie on the internet: Ignoring the privacy policies and terms of service policies of social networking services.

Obar, J. A., & Oeldorf-Hirsch, A. (2017). Clickwrap Impact: Quick-Join Options and Ignoring Privacy and Terms of Service Policies of Social Networking Services.

Oltramari, A., Piraviperumal, D., Schaub, F., Wilson, S., Cherivirala, S., Norton, T. B., Russel, N.C., Story, P., Reidenberg, J., Sadeh, N. (2017). PrivOnto: A semantic framework for the analysis of privacy policies. *Semantic Web,* (Preprint), 1-19.

Piacentini, M, Morgan, P, Miltner, J, Ravanini, R, Peinthor, R, Kleusberg, M, Clift, J & Haller, JT (2015). DB Browser for SQLite (Version 3.10.1) [Logiciel] Repéré à https://sqlitebrowser.org/

Pinterest (2018). Pinterest (Version 6.59.0) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.pinterest

Pinterest Inc. (2016). Privacy policy. Repéré à https://pinterest.com/about/privacy/

Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., & Chen, Z. (2014). Autocog: Measuring the description-to-permission fidelity in android applications. Dans *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1354-1365). ACM.

Rasthofer, S., Arzt, S., & Bodden, E. (2014). A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. Dans *NDSS*.

Rastogi, V., Qu, Z., McClurg, J., Cao, Y., & Chen, Y. (2015). Uranine: Real-time privacy leakage monitoring without system modification for android. Dans *International Conference on Security and Privacy in Communication Systems* (pp. 256-276). Springer.

Reidenberg, J. R., Breaux, T., Cranor, L. F., French, B., Grannis, A., Graves, J. T., Liu, F., McDonald, A., Norton, T. B., Ramanath, R. (2015). Disagreeable privacy policies: Mismatches between meaning and users' understanding. *Berkeley Tech. LJ, 30*, 39.

Ren, J., Rao, A., Lindorfer, M., Legout, A., & Choffnes, D. (2016). Recon: Revealing and controlling pii leaks in mobile network traffic. Dans *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services* (pp. 361-374). ACM.

Rovo89 (2017) Xposed development tutorial.
Repéré à https://github.com/rovo89/XposedBridge/wiki/Development-tutorial

Rovo89 & Tungstwenty (2011) XPosed (Version 89.2) [Logiciel] Repéré à https://forum.xda-developers.com/xposed/xposed-installer-versions-changelog-t2714053

Sadeh, N., Acquisti, A., Breaux, T. D., Cranor, L. F., McDonald, A. M., Reidenberg, J. R., Smith, N. A., Liu, F., Russel, N. C., Schaub, F. (2013). *The usable privacy policy project*. Technical report, Technical Report, CMU-ISR-13-119, Carnegie Mellon University.

Sathyendra, K. M., Wilson, S., Schaub, F., Zimmeck, S., & Sadeh, N. (2017). Identifying the Provision of Choices in Privacy Policy Text. Dans *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 2774-2779).

Shih, F., Liccardi, I., & Weitzner, D. (2015). Privacy tipping points in smartphones privacy preferences. Dans *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 807-816). ACM.

Shilton, K. (2009). Four billion little brothers?: Privacy, mobile phones, and ubiquitous data collection. *Communications of the ACM, 52*(11), 48-53.

Shklovski, I., Mainwaring, S. D., Skúladóttir, H. H., & Borgthorsson, H. (2014). Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. Dans *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2347-2356). ACM.

Skype (2018). Skype (Version 8.17.0.4) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.skype.raider

Slavin, R., Wang, X., Hosseini, M. B., Hester, J., Krishnan, R., Bhatia, J., Breaux, T. D., Niu, J. (2016). Toward a framework for detecting privacy policy violations in android application code. Dans *Proceedings of the 38th International Conference on Software Engineering* (pp. 25-36). ACM.

Snap Inc. (2018). Snapchat (Version 10.29.0.0) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.snapchat.android

Snap Inc. (2018). Snapchat Privacy Policy.
Repéré à https://www.snap.com/en-US/privacy/privacy-policy/

Song, Y., & Hengartner, U. (2015). Privacyguard: A vpn-based platform to detect information leakage on android devices. Dans *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 15-26). ACM.

Spensky, C., Stewart, J., Yerukhimovich, A., Shay, R., Trachtenberg, A., Housley, R., & Cunningham, R. K. (2016). SoK: privacy on mobile devices–it's complicated. *Proceedings on Privacy Enhancing Technologies, 2016*(3), 96-116.

Tan, J., Nguyen, K., Theodorides, M., Negrón-Arroyo, H., Thompson, C., Egelman, S., & Wagner, D. (2014). The effect of developer-specified explanations for permission requests on smartphone user behavior. Dans *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 91-100). ACM.

Thurm, S., & Kane, Y. I. (2010). Your apps are watching you. *The Wall Street Journal, 17*(1).

Twitter, Inc. (2017). Twitter Privacy Policy. Repéré à https://twitter.com/en/privacy

Twitter, Inc. (2018). Twitter (Version 7.27.0) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.twitter.android

Wang, H., Hong, J., & Guo, Y. (2015). Using text mining to infer the purpose of permission use in mobile apps. Dans *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 1107-1118). ACM.

We Are Social Ltd & Hootsuite Inc. (2017) Digital in 2017 : Global Overview [Présentation SlideShare]. Repéré à https://www.slideshare.net/wearesocialsg

We Are Social Ltd & Hootsuite Inc. (2017) Digital in 2017 : Northern America. [Présentation SlideShare]. Repéré à https://www.slideshare.net/wearesocialsg/digital-in-2017-northern-america

WhatsApp Inc. (2016). WhatsApp Legal Info.
Repéré à https://www.whatsapp.com/legal/#privacy-policy

WhatsApp Inc. (2018). WhatsApp Messenger (Version 2.18.79) [Logiciel]
Repéré à https://play.google.com/store/apps/details?id=com.whatsapp

Wijesekera, P., Baokar, A., Hosseini, A., Egelman, S., Wagner, D., & Beznosov, K. (2015). Android Permissions Remystified: A Field Study on Contextual Integrity. Dans *USENIX Security Symposium* (pp. 499-514).

Wilson, S., Schaub, F., Ramanath, R., Sadeh, N., Liu, F., Smith, N. A., & Liu, F. (2016). Crowdsourcing Annotations for Websites' Privacy Policies: Can It Really Work? Dans *Proceedings of the 25th International Conference on World Wide Web* (pp. 133-143). International World Wide Web Conferences Steering Committee.

Wu, J., Husson P.H., Dutta, K., Shanks, A., Verituse, Z. & Brindle, J. (2016) Magisk Manager (Version 16.0) [Logiciel] Repéré à https://magiskmanager.com/

Yu, L., Luo, X., Liu, X., & Zhang, T. (2016). Can we trust the privacy policies of android apps? Dans *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on* (pp. 538-549). IEEE.

Yu, L., Zhang, T., Luo, X., & Xue, L. (2015). Autoppg: Towards automatic generation of privacy policy for android applications. Dans *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 39-50). ACM.

Zheng, C., Zhu, S., Dai, S., Gu, G., Gong, X., Han, X., & Zou, W. (2012). Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. Dans *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices* (pp. 93-104). ACM.

Zhu, D. Y., Jung, J., Song, D., Kohno, T., & Wetherall, D. (2011). TaintEraser: Protecting sensitive data leaks using application-level taint tracking. *ACM SIGOPS Operating Systems Review, 45*(1), 142-154.

Zimmeck, S., Wang, Z., Zou, L., Iyengar, R., Liu, B., Schaub, F., Wilson, S., Sadeh, N., Bellovin, S.M., Reidenberg, J. (2017). Automated analysis of privacy requirements for mobile apps. Dans *Proceedings of the Network and Distributed System Security (NDSS) Symposium* (Vol. 2017).