Présentation de la sécurité sur android, modèles de menaces et stratégies de détection

Marwa Abdallah Mohamed Kamil, *DIRO,* Théophile Henri, *DIRO.*

***Abstract*—Au cours de ce module, nous avons vu les différents éléments qui composent la sécurité d'un système, répondant à des modèles de menaces aussi variés que potentiellement dangereux. Dans cet exposé, nous discuterons des problématiques de sécurités rencontrés sur android. Ce système d'exploitation, très largement utilisé, attire les développeurs malveillants qui y voient une niche de profits et rendent leur code de plus en plus complexe. Les chercheurs académiques et les compagnies vendant des antivirus doivent rapidement apprendre à adapter leurs méthodes à ces challenges, voire adopter des stratégies très différentes. Ce document a pour but de présenter l'état de l'art actuel dans ce contexte particulier.**

***Keywords*—Sécurité, Android**

I. Introduction

Les téléphones mobiles ont tranformé notre vie d'une manière que l'on aurait pas soupçonné il y a encore 15 ans.
En 1999, Nokia a dévoilé son premier téléphone capable de naviguer sur internet. Le Nokia 7110 utlisait le Wireless Application Protocole, ou WAP, version allégée du web permettant de consulter des pages textes, et de consulter ses consulter ses mails sur des sites compatibles. On dépasse largement les fonctionnalités classiques des téléphones mobiles d'appels, eux-mêmes technologie nouvelle et émergente. La réaction des concurrents ne s'est pas faite attendre, et Ericsson a annoncé la sortie du modèle R320S, lui aussi compatible avec le WAP, quelques mois plus tard.
Rapidement, les fournisseurs de service vont proposer des versions mobiles de leurs sites via le WAP.
Cette anecdote marque le début d'une nouvelle époque: celle de fréquentes spectaculaires innovations techniques, de la rapidité de leur démocratisation à l'échelle mondiale pour les consommateurs, et de la nécessité pour les entreprises spécialisées de les suivre pour adapter les services ou produits vendus.

Aujourd'hui, les smartphones, téléphone mobiles intelligents dont les fonctionnalités s'apparentent à celles d'un ordinateur représentent un marché très lucratif avec environ 300 millions de smartphones vendus chaque année. Dotés d'un processeur puissant, ils permettent d'avoir la plupart des fonctionnalités d'un ordinateur
Nous pouvons illustrer l'ubiquité des smartphones, et l'utilisation que l'on en fait dans un pays disposant d'un débit suffisant avec le cas des Etats-Unis.

Marwa Abdallah etc. est étudiante du DIRO, en provenance de l'université Lyon 1.

Théophile Henri est étudiant en échange en provenance de l'école d'ingénieur CPE Lyon situé sur le campus de l'université Lyon 1.

En 2015, 58% des américains possèdent un smartphone, contre seulement 35% en 2011[1]. En plus des applications classiques d'un téléphone qui sont la communication par appel et texte, 57% des américains font des opérations bancaires, près des 2/3 utilisent les fonctionnalités GPS pour être guidés dans leur déplacement. D'ailleurs, près de 7% des américains s'appuyent sur leur smartphone comme seul moyen personnel pour naviguer sur le web et effectuer les opérations courantes.
Android, système d'exploitation développé actuellement par Google s'est rapidement imposé par le nombre d'utilisateurs, compratívemenet a l'iOS d'Apple ou au windows phone de Microsoft. En 2016, Android représente 80% du marché. De cette popularité naît l'intérêt des développeurs malveillants. Ceux-ci vont chercher à attirer l'utilisateur à télécharger un malware, une application malveillante conçue de manière a faire des actions non légitimes, dans un but précis, souvent générer du profits. Dans le contexte des smartphone, la vulnérabilité naît de l'installation d'applications dans un hôte, un téléphone. Ensuite, elle peut se propager à l'intérieur de ce périphérique. Puisque les smartphones disposent de divers moyens et standards de communication (SMS, Bluetooth, 3G etc.), une infection peut ensuite se propager dans d'autres mobiles.
Pour notre exposé sur la sécurité d'android, nous allons expliquer le contenu d'une application, puis nous mentionnerons le fonctionnement du paiement sous android. Dans la section III, nous présenterons les formes de menaces rencontrés. La section IV va présenter les méthodes utilisées pour déterminer le caractère malveillant, ou non d'une application. C'est un problème difficile suscitant un grand intérêt. Enfin, nous parlerons d'approches novatrices pour finir par conclure.

II. L'architecture des applications et de la sécurité sous Android

Pour comprendre comment une application peut-être attaquée, il est nécessaire de dévoiler en détail la structure de celle-ci.

A. Les composants d'une application

Une application installée est enpaquetée dans une archive zip .apk contenant plusieurs fichiers et dossiers. En particulier, le fichier *AndroidManifest.xml* contient les informations sur les permissions requises par l'application et la définition des composants de l'application, entre autres.

Les composants d'une application Androïd définis dans le manifeste sont un ensemble d'un ou plusieurs de chacun des élément décrits ci-dessous:

- L'**Activité** est la partie interface homme-machine de l'application, une activité par écran présenté à l'utilisateur. C'est donc le point d'entrée de l'application. Par exemple, une application mail pourrait avoir une activité qui présente la liste des emails non-lus, une autre pour la composition d'un email et une autre pour la lecture d'un email.
- Le **Service** exécute des tâches en arrière plan, souvent pour une longue période, sans intéraction directe avec l'utilisateur. On peut illustrer avec le fait que la tâche "jouer la musique" peut s'éxecuter alors que l'utilisateur est sur une autre application.
- Le **Receveur Broadcast** est à l'écoute des évents générés par le système, comme par exemple la réception d'un SMS. Un application peut également recevoir des évents envoyés par d'autres applications de l'appareil en broadcast. Ces évènements seront traités par le composant service.
- Le **Content Provider** est chargé d'aider l'application à gérer l'accès à ses données, aux données d'autres applications. Il offre des mécanismes de sécurisation des données.

B. La sécurité en amont

Elle est définie par différents mécanismes, opérants à différents niveaux. Dans ce qui suit, nous allons les présenter en commençant au niveau du marché, puis au niveau de la plateforme, sur l'appareil utilisateur.

1) Au niveau du marché: Google propose la boutique Google Play à ses utilisateurs. Cette dernière, préinstallée dans les appareils, contient dans son marché des applications développées par Google, et par des développeurs tiers. En plus, Google permet aux utilisateurs d'Android de télécharger des applications disponibles sur d'autres marchés que Google Play, contrairement à iOS qui ne permet à ses utilisateurs de télécharger des applications qu'à partir de la boutique officielle, l'App Store.
En 2012, Google a introduit Bouncer, une solution scannant les applications disponibles dans le Play Store a la recherche de malware. Les applications sont étudiées avant leur mise à disposition pour les utilisateurs, en étant téléchargées dans un pot de miel simulant un mobile utilisateur.
Certains marchés tiers proposent des dispositifs de contrôle comme l'Amazon App Store Distribution Portal, processus de validation employé sur le marché tiers d'Amazon lancé pour concurrencer le Play Store. D'autres marchés ne concoivent pas la sécurité en amont.
Par ailleurs, chaque application doit-être signée et auto-certifiée par le développeur, qui est informé de sa responsabilité de choisir une clé privée sécuritaire et de la garder privée. La signature sera brisée si l'application a été modifié. Cela sécurise aussi le processus de mise à jour, car côté hôte, seules les mises à jour signées par le développeur légitimes seront installées.

2) Au niveau de la plateforme: Il y a différents niveaux de sécurité. Le développeur d'une application légitime voudrait pouvoir protéger son produit installé dans un environnement possiblement infecté. Il ne faut pas le code malveillant entre dans l'application. Parrallèlement, il ne faut pas qu'une application malveillante puisse accéder à d'autres applications et au système.

Le ***sandboxing*** est un mur virtuel entre les applications, au niveau OS, permettant de limiter les interactions qu'elles pourraient avoir entre elles, et avec le système. Concrètement, chaque application se voit attribuer un identifiant unique, l'UID, et sera exécutée dans un processus isolé.
Si deux applications sont proposées, signées par le même développeur, Androïd leur attribuera de facto le même UID.
En outre, les applications peuvent-être regroupées dans un ou plusieurs groupes, chacun identifié par un GID. L'ensemble des applications d'un groupe partageraient un droit, comme l'accès en lecture/écriture à la carté mémoire SD où la possibilité d'utiliser bluetooth.
En bref, isoler une application ou restreindre l'accès à des composants de l'hôte prévient l'intéraction malveillante, sans avoir à analyser cette dernière. La philosophie est un peu la même que celle du pare-feu.

Si le mécanisme d'isolation est primordial pour prévenir en partie les infestations et leur propagation dans un système, il ne doit pas être une fin en soi car il nuirait à la créativité des développeurs. On peut vouloir accéder à l'activité liée à l'application Android Appareil Photo, à partir d'une application de réseautage social (ex: facebook).
C'est là que les ***permissions*** interviennent. Elles sont une brique de sécurité importante opérant dans la couche échanges entre composants.

Nous distinguerons les permissions *demandées* par une application pour accéder à des ressources qui ne lui sont pas propres, des permissions que l'application *accorde* pour permettre à d'autres applications d'accéder à ses composants propres. Nous commencerons par le premier.

Pour avoir accès à des fonctionnalités sensibles du téléphone, comme les contacts ou la location GPS, un développeur doit nécessairement en faire la demande explicite dans le fichier *AndroidManifest.xml*. Cette permission peut-être accordée automatiquement par le système ou doit-être demandée à l'utilisateur. Elles sont divisées dans les 4 niveaux de protections décrits ci-dessous

- **Normal:** accorder ces permissions ne présente pas un grand risque pour le système, elles sont automatiquement acceptées (ex: vibreur).
- **Dangerous:** donne l'accès à des informations confidentielles de l'utilisateur, comme sa liste de contacts. Il en incombe à l'utilisateur d'accorder la permission.
- **Signature:** Ces permissions ne sont accordées que si les applications concernées ont été signées par le même développeur.
- **SignatureOrSystem:** Android ne donne cette permission qu'aux applications dans l'Android system image ou à celles vues dans Signature. Un exemple est

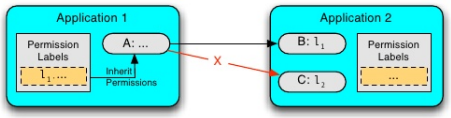

Fig. 1. Communication inter composants

set_time. Il est conseillé de ne pas demander cette permission.

Depuis 2015 et Android 6.0, la demande des permissions ne se fait plus au moment de l'installation, mais pendant l'éxecution au moment où l'application en a besoin. Une permission refusée par l'utilisateur entraîne la non exécution par l'application des tâches qui en avaient besoin. Le fait qu'un utilisateur ne discerne pas forcément la nécessité d'accorder une permission de la demande accessoire peut ouvrir la porte de l'exploitation.

D'autre part, une application peut vouloir partager ses fonctionnalités avec une autre. Le développeur devra explicitement associer une permission sous forme de label qu'il définira à un composant dans *AndroidManifest.xml* pour permettre à une, plusieurs, où toutes les applications d'y accéder. Ces permissions sont assujetties à la même classification que nous avons vu dans le paragraphe précédent.

Le moniteur de référence, situé dans la couche middleware d'Android, fournit une méthode de gestion d'accès type mandatory access control, MAC. Dans ce type de gestion d'accès, les décisions sont centralisés et les utilisateurs se voient attribuer des droits qu'ils ne peuvent modifier (par opposition au DAC qui met en avant le concept attribué à un utilisateur lui conférant le droit de gestion des accès). Pour illustrer, le modèle Bell LaPadula implémente MAC pour L'attribution des droits aux sujets et aux objets, non modifiables par ceux-ci quelque soit leur niveau de sécurité.

Au moment de l'installation de l'application, un ensemble de permissions lui sont assignés de manière statique par Android, qui va aussi évaluer les permissions demandées.

Lorsqu'une demande de communication inter-composants (ICC) est initiée, le moniteur de référence évalue l'APL de l'émetteur de la demande. Si le composant destinaire a le même APL défini, la communication est initiée, sinon elle est refusée, même à l'intérieur d'une même application.

La figure 1 montre un exemple de communication. Le composant A de l'application 1 hérite des permissions de cette dernière. Ici, la communication est intiée avec le composant B, mais pas avec le composant C de l'application 2, car ces deux dernières ne partagent pas les mêmes APL. Cela est possible si le développeur a spécifié de refuser l'accès au composant C pour le protéger, tout en laissant les autres composants, comme B, hériter du caractère accessible.

III. Le paiement sous Android

Il est courant pour les applications Android d'utiliser le paiement in-app pour fournir des améliorations de contenue aux utilisateurs, dans cette partie, nous allons voir les composantes principales nécessaire à la mise en place d'un paiement in-app.

Les applications ont accès au paiement in-app à l'aide d'une API qui est contenue dans l'application google play de l'appareil: In-app Billing API. Tout développeur peut intégrer ce service dans leur application sans recourir a un contrat avec Google. Les seules conditions sont *a)* posséder un compte développeur dans Google play et *b)* un compte dans Google Wallet marchand.

En pratique, l'application ne communique jamais avec les serveur google play, elle envoie des requêtes de paiement a l'application Google Play à l'aide des de communication interprocessus (IPC). L'application ne gère aucune connection réseau entre elle et les serveur google play.

L'API In-app Billing (IAB) propose entre autres les fonctionnalités suivantes :

- L'application envoyer des requêtes permettant l'achat de produits in-app.
- L'API propage les informations de commandes a l'appareil une fois l'achat complété.
- Tous les achats sont surveillés. L'utilisateur ne peut posséder plusieurs copies du même achat in-app.
- L'API supporte un système d'abonnement.

Un achat à l'aide de l'API se passe de la manière suivante :

- L'application envoie une requete à Google Play pour savoir si la version de l'API que l'on utilise est supportée.
- Quand l'application démarre et l'utilisateur se connecte, on vérifie avec Google Play ce que possède déjà l'utilisateur. Google Play renvoie une liste des identifiants des produits achetés, une liste des détails d'achats et une liste des signatures pour ces achats.
- Google Play informe l'utilisateur des achats qui sont disponibles en lui renvoyant une liste contenant les détails des produits comme le prix, le titre, la description et le type d'achat.
- Si un produit n'est pas possédé par l'utilisateur, on peut envoyer une requête d'achat qui spécifie l'identifiant du produit ainsi que d'autres paramètres. Google Play répond si l'achat a été un succès ou annulé. Si l'achat est un succès, la réponse contient une chaine de caractères qui identifie de façon unique chaque transaction et la signature de l'achat, signée avec la clé privée du développeur.

Attaques sur IAB

Un certain nombre d'attaques ont été pensées par la communauté. En 2011, il a été montré possible dans une preuve de concept d'effectuer les achats sans payer[5]. Dans le protocole IAB, Google signe un message notifiant une application que la transaction a bien été effectué. L'application concernée doit vérifier la signature, avant d'envoyer un accusé de réception à Google. Les développeurs sont informés de la necessité de cette vérification mais ne l'incorporent

pas tous dans leur code, ou effectue la vérification de signature dans le périphérique en utilisant une API java: java.security.Signature.verify. Les chercheurs de l'université de Berkeley ont réecrit les applications, et changé tous les appels à java.security.Signature.verify avec une méthode qui retourne toujours True. Cette attaque a été effective pour la plupart des applications réecrites. Celles qui ont su se protéger, le plus souvent a la base des applications qui encaissent de plus grand montantsm, effectue des vérifications plus poussées de signature. Ici, la responsabilité revient aux développeurs de connaître les vulnérabilités, d'être à jour et de sécuriser leurs applications en conséquence.

IV. Menaces et moyens de propagations

Dans cette partie, nous allons voir les principales menaces dont est sujet Android et les principaux moyens qu'elles utilisent pour se propager.

A. Menaces

- Attaque par augmentation de privilège : cette attaque consiste à exploiter une vulnérabilité du kernel Android disponible publiquement pour obtenir un acces root à l'appareil.

- Fuite d'information personnelle ou vol d'information personnelle : cette attaque se produit quand l'utilisateur donne de dangereuses permissions à une application qui, sans que l'utilisateur le sache ni ne le veuille, peut accéder à des données privées ou importantes puis les partager.

- Certaines applications malhonnêtes peuvent espionner l'utilisateur en surveillant les appels, les SMS/MMS, les opérations bancaires, ou encore enregistrer des vidéos sans que l'utilisateur le sache.

- Certaines applications malhonnêtes peuvent gagner de l'argent en effectuant des appels ou en s'abonnant à des services premium à l'aide de SMS sans que l'utilisateur le sache.

- Certains virus permettent aux adversaires de controler l'appareil à distance à l'aide d'un serveur en lui envoyant divers commande, dans le but d'effectuer des activités malhonnêtes.

- Certaines campagnes publicitaires agressives obligent l'utilisateur à télécharger des applications malhonnêtes, ou même des virus.

- Attaque par complot d'applications : cette attaque se produit quand un ensemble d'applications signés avec le même certificat sont installés sur un appareil. Le principe est que toutes ces applications vont partager entre elles leurs différentes permissions, afin d'avoir une application malhonnête ensemble. Par exemple, si une application possède le droit SMS_READ et une autre application possède le droit internet, la première peut obtenir des données sensibles de l'utilisateur à travers de la lecture des SMS, puis elle peut envoyer ces informations à l'autre application qui pourra les exporter à l'aide de internet.

- Attaque par déni de service (DoS) : le principe est de limiter la capacité de fonctionnement du téléphone, par exemple quand des applications surutilisent des ressources limitées comme le CPU, la mémoire, la batterie ou la bande passante. Ceci a pour conséquence que l'utilisateur ne pourra utiliser son appareil que d'une façon restreinte.

B. Principaux virus d'Android

- Trojan : les trojans se font passer pour des applications classique, cependant, celles-ci permettent des activités malhonnêtes sans le consentement ou le savoir de l'utilisateur. Ils permettent la fuite d'informations confidentielles de l'utilisateur ou peuvent meme utiliser l'ingénierie sociale pour obtenir d'autres informations comme des mots de passe. Par exemple, il existe des trojans d'application SMS, qui peut envoyer des SMS à des services payants, il existe également un trojan demandant d'entrer ses identifiants netflix, afin que l'adversaire puisse les récupérer.

- Backdoor : Les backdoor permettent à d'autre virus d'entrer dans le système de façon discrète en contournant les méthodes de sécurité. Par exemple, une méthode est d'utiliser le root pour gagner les privilèges de super utilisateur et se cacher des scanner antivirus.

- Worm : les worm peuvent se propager en créant des exactes copies d'eux-mêmes puis en utilisant les différents systèmes de communication. Par exemple, les worm bluetooth peuvent utiliser les fonctionnalités bluetooth de l'appareil pour se propager à tout les appareils appairés.

- Botnet : les applications Botnet compromettent l'appareil en créant un bot, ce bot permet à un utilisateur malhonnête de controller l'appareil à l'aide d'un serveur distant, appelé bot-master, à l'aide de différentes commandes. Le terme Botnet correspond à un réseau d'un ensemble de bot. Les applications possibles pour un Botnet peuvent être l'envoie d'informations privée, l'utilisation des fonctionnalités du téléphone pour entrainer une attaque par déni de service ou encore l'installation d'applications payantes.

- Spyware : les applications Spyware permettent d'espionner un utilisateur en surveillant ses messages, ses appels, sa localisation ou encore ses données bancaires. Le Spyware peut ensuite envoyer les informations collectées à un serveur.

- Agressive Adware : ces virus peuvent créer des raccourcies sur l'écran d'accueil de l'appareil, changer le moteur de recherche par défaut ou encore envoyer des notifications inutiles afin de diminuer les performances de l'appareil.

- Ransomware : les ransomware bloquent un appareil ainsi que ses données, le rendant inutilisable, l'adversaire demande ensuite une certaine somme d'argent pour débloquer l'appareil.

C. Moyens de propagation

- Repackaging d'application populaire : le principe est de décompiler une application du store officiel, puis d'insérer le malware que l'on souhaite injecter, réassembler l'application trojan puis la redistribuer à l'aide d'app store locaux moins surveillés. Différents outils sont disponibles pour désassembler une application. Cette méthode est l'une des méthodes de génération de virus les plus utilisées.

- Drive-by-download : un adversaire peut utiliser le social engineering, aggressive advertisement ou le click sur une URL malhonnête pour forcer l'utilisateur à installer une application malhonnéte. De plus, cette méthode doit déguiser une application malhonnête en une application légitime pour inciter l'utilisateur à installer l'application.

- Dynamic payload : les applications peuvent embarquer des taches malhonnêtes en tant qu'exécutable apk/jar, soit encrypté ou en texte clair à l'intérieur de l'APK source. Une fois installée, l'application décrypte l'exécutable. Si l'exécutable est un jar, le virus charge l'API DexClassLoader et exécute le code dynamiquement. Si l'exécutable est un apk, le virus peut influencer l'utilisateur à l'installer en indiquant une mise à jour nécessaire. Certain virus de ce type n'embarquent pas directement l'exécutable malhonnête mais le télécharge à partir d'un serveur pour éviter la détection de virus.

- Stealth malware technique : Android est développé pour des appareils limités en ressources comme les téléphones portables, qui ont donc des batteries limitées ainsi que des processeurs ou une mémoire plus faible qu'un ordinateur. Les antivirus ne peuvent donc pas procéder à une analyse profonde en temps réel du code des applications, au contraire des antivirus sur ordinateur. Les créateurs de virus utilisent donc cette limitation pour déguiser leur virus à l'aide de chiffrement du code, de permutations de clé, de chargement dynamique du code.

V. Un exemple de malware

A l'approche du SuperBowl 2012, le malware "MADDEN_NFL_12_1.0.3" a été découvert. Il imite une application légitime dévelloppée par la célèbre est reconnue EA Games, "Madden NFL 12" dont la version à l'époque était 1.0.3. Le malware était mis à disposition dans un marché tiers, et l'icone étaoit la même que celle de l'application légitime, pour mieux jouer sur la psychologie des utilisateurs. Son but est le gain d'argent par envoi de SMS à des numéros premium.
Uns fois éxécutée, l'application affiche "Hello World, AndroidMeActivity!", et fait tourner son code malveillant en arrière plan.
3 composants présentés comme des images sont infectés. Ce sont en réalité des éxécutables. *header01.png* permet d'acquérir les droits root, *border01.png* est un cheval de Troie SMS et enfin *footer01.png* est un IRCBot, pour se connecter à des salons de chat en utilisant le protocole de communication IRC.
Le déroulement est le suivant: une fois les droits root acquis par élévation de privilèges, le dévelopeur peut éxecuter les commandes à partir d'un serveur distant. Ceci est permit par Android pour des utilisations légitimes comme donner à un utilisateur le contrôle de son téléphone depuis son PC. L'IRCBot sert au contrôl distant: Ce backdoor génère un login aléatoire pour se connecter au chatroom désiré. De là, le malware va attendre et exécuter les ordres reçus.

En utilisant la position géographique, des SMS surtaxés sont envoyés à des numéros premium. De plus, si un message est recu pendant que le malware est éxécuté, celui-ci vérifie l'origine du message. S'il provient d'un numéro surtaxé, il est détruit. Sinon, le numéro de téléphone et le contenu du message de l'émetteur est dévoilé au serveur distant. Cela peut-être d'autant plus compromettant si la victime recpoit un message contenant des infornations sensibles comme un message de sa banque révélant des informations confidentielles comme le montant du compte courant.

VI. Evaluation, analyse et détection des malware

La définition de la meilleure approche de détection d'intrusion est le nerf de la guerre. Nous allons donc commencer par nous intéresser aux approches de détection et d'analyse de malware. Ensuite, nous présenterons les outils mis à disposition, par Google et d'autres compagnies.

A. Les méthodologies

Pour faire façe à la profiférations des malwares, il faut-être capable de reconnaître les comportements anormaux, voire suspicieux.
Il existe 2 grand types d'approches de détection:

1) L'approche statique: Elle consiste à l'analyse du code de l'application, sans l'éxecuter. C'est une méthode rapide, mais sans éxecution elle est inefficace contre les applications qui obscurcissent le contenu malveillant, le plus souvent en l'encryptant.

le type *signature-based*, très utilisé par les anti-virus, repose sur le fait que chaque malware rencontré est repertorié dans une base de données. Chaque virus connu est haché, en utilisant MD5 par exemple. Une signature de virus, un fingerprint unique est crée. (il existe quelques rares collisions de 2 malwares vers un même hash mais en pratique on modifiera

un peu le résultat). Le hash généré est stocké dans la base de données, et est partagé dans la communauté. Cela permet de reconnaître un virus non modifié.
Les développeurs malveillants l'ont bien compris et tentent de tromper les antivirus en rendant leur code polymorphique pour être indétectable. Les compagnies vendant les antivirus répliquent en faisant des analyses statiques heuristiques. C'est à dire qu'elles vont scanner les fichiers de l'application à la recherche de features suspectes, telles certaines librairies importées, ou l'analyse des fichiers binaires à la recherche de code compressé.

2) l'approche dynamique: Elle va exécuter le code dans un environnement protégé, et inspecter les intéractions avec le système et le comportement général de l'application. Ici, un environnement Android est émulé, de l'architecture aux fonctionnement. On va déclencher les évents tels le swipe ou les entrées clavier ainsi que les évents systèmes normaux rencontrés lorsqu'une application est téléchargée et installée par un utilisateur. En particulier, Les approches basées sur le comportement, ou *behavior-based* modélise le comportement normal que l'on attend, et classifient comme anormales tout autre comportement obervé. Ces approches ont l'avantage de pouvoir détecter un zero-day malware, autrement dit un malware non répertorié. .L'inconvénient est le haut taux de faux positifs, c'est à dire les comportements légitimes classifiés comme malicieux.

Par exemple, on peut détecter de l'information disclosure sur des données privées ou l'envoi de SMS effectué sans le consentement de l'utilisateur.
Si cette approche est plus efficace contre les malwares polymorphes, l'inconvénient majeur est que certaines éxécutions malicieuses peuvent passer sous le radar. Le comportement utilisateur simulé est généré de manière aléatoire, ce qui peut ne pas déclencher l'exécution du code malicieux.

3) Rise of the machine...learning: Comme dans beaucoup d'autres disciplines de l'informatique, il peut-être très intéressant de chercher à utiliser l'apprentissage automatique dans les données massives collectées à chaque fois que nous rencontrons une nouvelle forme de malware pour prédire les futures attaques.
Les études créent des datasets à partir d'applicatioons bénignes et malveillantes. La première phase consiste en l'extraction des features. Ils varient dans la littérature d'informations sur l'état du système (consommation mémoire, batterie...) lorsqu'on utilise une approche dynamique, ou sur la syntaxe du code statique (fichiers librairies, fichiers jar...). L'ensemble d'entraînement est composé de dizaine de miliers d'exemples dont on sait la classification réelle, bénin ou malicieux. On peut alors entraîner des algorithmes de classification supervisés, comme SVM, K plus proches voisins par exemple, et tester les paramètres appris sur un ensemble dédié. Comme dans toutes les propositions de modèles de détection d'intrusion, le but est de créer un modèle minimisant le plus possible les faux positifs et les faux négatifs. La recherche de cet équilibre passera par l'entraînement constant dans des datasets de plus en plus grands et enrichis des nouvelles formes de malwares.

Les résultats des classifieurs sont prometteurs. Pour illustrer, nous nous permettons de faire une digression et de présenter une étude qui n'a pas porté sur les applications android, mais de fichiers infectés (provennant de bases de données de sites spécialisés) et bénins (récoltés dans les fichiers exécutables de leur ordinateur). La philosophie est la même et notre but est purement pédagogique. Ainsi, Santos et Al[10]. utilisent les opcodes, instructions en code assembleur comme *push* ou *mov*, qui peuvent aider à débusquer le code polymorphique. Une étude précédente faite par Bilar en 2007[11] a montré que les fichiers contiennent des distributions différentes de ces commandes selon qu'ils soient obfusqués ou non. Les auteurs ont utilisé les commandes assembleurs commes attributs X, et la classification Y est malware ou bénin. Le but est de déterminer des associations de commandes assembleurs prédisant de manière sûre un type d'application ($X_{commande1}$, $X_{commande5}$ et $X_{commande13} => Y_{Malware}$ par exemple). Leur dataset contient environ 26000 exemples répartis équitablement entre codes légitimes et chevaux de Troie. Les algorithmes utilisés (K plus proches voisins,fôrets aléatoires, SVM et réseaux bayésiens) ont montré des taux d'accuracy entre 90 et 95%. Des résultats prometteurs pour une approche statique.

B. Les techniques de déploiement

Les méthodes que nous venons de voir peuvent-être déployées dans des supports différents, en fonction des ressources nécessaires pour procéder à l'analyse.

1) Dans le téléphone: Dans les cas des approches basées sur la signature. Elle couvre un grand nombre de malwares. Les applications anti-malware en sont un exemple. Par exemple, Avast Mobile Security est un antivirus gratuit disponible dans Play Store. Si elle est paramétrée pour être éxecutée pendant la pahse d'installation d'une nouvelle application, l'archive .apk est scannée avant son installation. Si une menace est détectée, Avast propose à l'utilisateur de désinstaller l'application. Cependant, les déploiements sur le périphérique sont sujets au sandboxing et ne peuvent scanner l'ensemble des fichiers, notamment les fichiers privés d'autres applications. Sans privilèges roots, leurs actions après détection sont aussi limitées, l'accord explicite de l'utilisateur étant nécessaire.

2) Distribué: Pour palier au problème de ressources, une partie de l'analyse et de la détection peut-être exécutée dans le mobile, et le travail le plus coûteux peut-être effectué sur un serveur distant. Une analyse basée sur le comportement d'une application pourrait éxecuter les tâches, récolter un certain nombre de résultats d'éxecution et envoyer à un serveur pour analyse.

3) Dans un cloud: Un serveur peut contenir plusieurs environnements virtuels simulants autant de plateformes sous Android. Ces pots de miel vont télécharger les applications et effectuer des analyses en profondeur avec d'importantes ressources mémoires et puissance de calcul. C'est le mode utilisé par Bouncer qui a choisi l'approche dynamique pour la détection.

Les solutions proposées, sont en général dans le milieu académique, ou sont distribuées à travers des applications anti-malware. Si Google a su prévenir 40% des malwares sur

Play store avec Bouncer, et a intensifier la protection dans le terminal avec les mécanismes vus dans la section II, un effort colossal doit-être fait pour déployer à grande échelle une ou plusieurs solutions protégeant à les utilisateurs du marché au terminal.

VII. Futures directions

L'intelligence artificielle promet d'être un outil incontournable pour la détection de malware sophistiqués. Les méthodes d'apprentissage machine supervisés que nous avons vu ont un succès isolé. Les auteurs choisissent des features souvent différents de manière non optimale car c'est un problème difficile. La sélection des features intéressant à miner est un sujet de recherche crucial.
A cette fin, l'apprentissage en profondeur non supervisé pourrait permettre de découvrir les features pertinents, dans une phase de pré-entraînement. La start-up Deep Instinct, élue compagnie la plus innovante en cybersécurité en 2016, est spécialisée dans la prévention des malwares zero-day, et des plus sophistiqués qui savent s'évader des approches de détection. C'est la première compagnie qui sort du cadre de la recherche pour proposer en pratique une approche applicable utilisant l'apprentissage en profondeur.

VIII. Conclusion

Au fil des années nous avons été témoin de l'évolution des formes de malwares et des techniques pour s'en prévenir.
Cela a commencé avec les signatures, utilisant le savoir déjà développé pour les ordinateurs. L'étape suivante a été l'heuristique, qui tente d'identifier les malwares non répertoriés a partir des caractéristiques de leur comportement. L'apprentissage machine, puis l'apprentissage en profondeur se sont montré prometteur pour permettre de développer des solutions de détection capables de reconnaître toute forme de malware jamais rencontré, mais utilisant forcément des méthodes déjà rencontrées.
A l'échelle d'un utilisateur ou d'une entreprise, le plus important reste la prévention. Les techniques de prévention sont inutiles si elles ne sont pas connues et ne sont pas employées. L'effort éducatif n'est pas négligeable.

References

[1] A. Smith, *U.S. Smartphone Use in 2015*, rapport pour Pew Research Center. Avril 2015, Disponible sur:http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015.

[2] Faruki et Al, *Android Security: A Survey of Issues, Malware Penetration, and Defenses*. IEEE communication surveys tutorials, Vol. 17, No. 2, second quarter 2015.

[3] Suarez-Tangil et Al, *Evolution, Detection and Analysis of Malware for Smart Devices*. IEEE communication surveys tutorials, Vol. 16, No. 2 VOL. 16, NO. 2, second quarter 2014.

[4] Eleanor Cawthon and Ben Zhang, *Android Security*. Université Berkeley, Californie. Nov 2015, disponible sur https://people.eecs.berkeley.edu/ raluca/cs261-f15/readings/android.pdf

[5] Reynaud et Al.,*FreeMarket: Shopping for free in Android applications*. University of California, Berkeley, 2011

[6] Mcafee Lab, *Evolution of Android Malware: IRCBot Joins the Party*, 2012. Disponible sur https://securingtomorrow.mcafee.com/mcafee-labs/evolution-of-android-malware-ircbot-for-android/.

[7] Treadwell et Zhou, *a heuristic approach for detection of obfuscated malware*. IEEE, ISI 2009, June 8-11, 2009, Richardson, TX, USA

[8] Ryo Sato1, Daiki Chiba2 and Shigeki Goto, *Detecting Android Malware by Analyzing Manifest Files* Proceedings of the Asia-Pacific Advanced Network 2013 v. 36, p. 23-31.

[9] Idika, Nwokedi and Aditya P. Mathur. *A Survey of Malware Detection Techniques*. (2007) Department of Computer Science Purdue University, West Lafayette.

[10] Santos et AL, *Opcode sequences as representation of executables for data-mining-based unknown malware detection*. Information Sciences Volume 231, 10 Mai 2013, Pages 64–82.

[11] Bilar, *Opcodes as predictor for malware*. International Journal of Electronic Security and Digital Forensics 1(2) · Janvier 2007.

[12] Arslan, Gunduz, Sagiroglu, *A review on mobile threats and machine learning based detection approaches*. Digital Forensic and Security (ISDFS), 2016 4th International Symposium on.

[13] Wolff, Davis, *Deep Learning on Disassembly Data*. Support de conférence pour BlackHat USA 2015. Disponible sur https://www.blackhat.com/docs/us-15/materials/us-15-Davis-Deep-Learning-On-Disassembly.pdf et aussi sur youtube

[14] Hardy et Al, *A Deep Learning Framework for Intelligent Malware Detection*. 2016, Department of Computer Science and Electrical Engineering West Virginia University, Morgantown, WV 26506, USA.

[15] https://developer.android.com/google/play/billing/billing_overview.html

[16] William Enck , Damien Octeau , Patrick McDaniel , Swarat Chaudhuri, *A study of android application security*, Proceedings of the 20th USENIX conference on Security, p.21-21, August 08-12, 2011, San Francisco, CA