La sécurité sous Android

Est-ce qu'Android est sécurisé ? Faut-il croire les bulletins publiés par les vendeurs d'Anti-Virus ? Quelles sont les solutions proposées par Google pour renforcer la sécurité ?

Par Philippe PRADOS - 2014
www.prados.fr

Depuis le début de la conception des OS pour mobiles, la sécurité est très présente. Contrairement aux OS plus anciens, l'isolation n'est pas limitée aux comptes utilisateurs. Chaque application est dans un bac à sable l'isolant des autres applications. En effet, depuis de nombreuses années, les virus et autres vers exploitent la moindre parcelle de privilège pour en abuser. Comme les téléphones sont particulièrement sensibles, il est important de les protéger. Une application malveillante peut récupérer des informations très personnelles, ou déclencher des appels ou SMS surtaxé, afin de monétiser les attaques.

Android propose différents niveaux de protections. Cela va de l'OS au Play store. Nous allons regarder tout cela pour évaluer le risque réel.

Niveau 1 : le boot

Le boot d'un système Android s'effectue en plusieurs étapes. La ROM du terminal prend la main, et décide de la partition à lancer. Suivant les paramètres, la ROM peut vérifier la signature de la partition avant le lancement. Ainsi, il n'est pas possible de modifier le système au démarrage. Ensuite, une chaîne de signature est mise en place pour vérifier la signature du noyau, et ainsi de suite. Cela utilise la technologie dm-verity qui construit un arbre de hash sur les secteurs de l'OS pour interdire leurs corruptions.

Android offre néanmoins, sur la plupart des téléphones, le moyen de désactiver ce verrou. Cela se traduit par un effacement complet du téléphone. Ainsi, il n'est pas possible d'alléger les privilèges pour pouvoir accéder aux données d'un téléphone protégé.

Niveau 2 : le chiffrement du terminal

Par défaut jusqu'à Android L, les données du téléphone ne sont pas chiffrées. L'utilisateur peut activer le chiffrement s'il le souhaite.

Concrètement, une fois le chiffrement activé, l'OS boot une première fois sur une partition non chiffrée, afin de pouvoir demander à l'utilisateur le secret de déverrouillage. Avec ce précieux, le système boot à nouveau avec la partition `/data` chiffré via `dm-crypt` et une clé 128 AES avec CBC et ESSIV:SHA256. Chaque accès à un secteur du disque sera chiffré et déchiffré.

Pour simplifier l'interface utilisateur, l'utilisateur doit utiliser le même mot de passe pour le chiffrement que pour verrouiller l'écran. Ce mot de passe doit être numérique et suffisamment long. En fait, ce mot de passe n'est pas réellement le secret de chiffrement du disque. Il sert à déchiffrer la clef de 128 bits. Ainsi, il est possible de modifier le verrou, cela chiffre à nouveau la clef. Il n'est pas nécessaire de chiffrer l'intégralité du disque à nouveau.

Du point de vue de l'utilisateur, utiliser un verrou complexe lors du lancement du téléphone est une bonne chose, mais pas pour une utilisation quotidienne. On aimerait avoir un mot de passe fort pour le disque, et un schéma plus simple pour l'écran. On trouve des applications dans le Play Store, s'exécutant sous Root, permettant de dissocier le verrou de chiffrement du

verrou de l'écran.

Après l'affaire Snowden, Google a annoncé que Android L imposera le chiffrement du disque du téléphone. Comme chez Apple, l'idée est de rendre impossible la récupération de données sur les téléphones, même si un juge le demande. À l'impossible, nul n'est tenu.

Il n’est plus nécessaire de saisir un mot de passe pour déverrouiller le chiffrement. Ce dernier est maintenant placé dans un conteneur sécurisé en hardware.

Niveau 3 : l'OS

Le système Android est dérivé de Linux. Il possède ainsi toute l'expérience de ce dernier. Les piles IP et autres sockets sont ainsi protégés par un code régulièrement audité.

Depuis la version 4.2, de gros effort ont été entrepris pour renforcer la sécurité à tous les niveaux.

Afin de renforcer la sécurité du noyau, le kernel propose maintenant une version enrichie de SELinux (proposé par un gars de la NSA, mais les sources sont ouvert). Chaque processus possède une liste de « capability ». Il peut en perdre mais pas en gagner. Cette couche logicielle permet de limiter au maximum les privilèges accordés à **root**. Chaque programme sous `root` n'a plus un accès complet au système. Dans un premier temps, ce système ne faisait que logger les violations (Android 4.3). Maintenant, il est activé (Android 4.4). L'objectif de ce composant est de circoncire un attaquant, s'il arrive à devenir `root` en exploitant une vulnérabilité d'un des programmes tournant sous ce privilège. Le code est spécifique Android pour renforcer les couches de communications entre applications Android via le Binder, un RPC interne.

D'ailleurs, les dernières versions d'Android réduisent radicalement le nombre de programmes tournant sous `root`. Par exemple, le démon d'installation `installd` n'a plus besoin de `root`.

Pour réduire le risque d'exploitation de bug dans la gestion de la mémoire, Android est maintenant intégralement compilé avec `FORTIFY_SOURCE`. C'est un paramètre et une librairie du compilateur permettant de détecter une partie des buffers overflows, les débordements lors des manipulations des chaînes de caractères (`strchr()`, `strrchr()`, `strlen()` et `umask()`) . Android 4.4 utilise la version 2 de `FORTIFY_SOURCE` qui renforce encore la sécurité.

Pour éviter les prédictions d'adresse mémoire, pour des attaques du type return-to-libc, Android est compilé avec ASLR. Cela permet de variabiliser les adresses mémoires du code et des données. À chaque lancement du programme, il utilise une adresse différente non prévisible.

Cette protection est à nuancer sous Android. En effet, pour éviter de lancer une nouvelle instance de Dalvik, la machine virtuelle d'Android, le système utilise un mécanisme de `fork()`. Un démon appelé Zygote est lancé par le script d'init de l'OS. Ce démon commence par charger toutes les classes du framework en mémoire. Puis il lance le garbage collector afin d'avoir une image du programme la plus stable possible. Enfin, Zygote attend des ordres venant d'un socket local à l'OS. Lorsque Android souhaite lancer une application, il envoie un ordre à Zygote avec le répertoire de l'application et la classe à lancer. Zygote commence par se dupliquer. Une nouvelle instance est alors présente, mais elle partage l'essentielle de la mémoire avec Zygote. Comme la mémoire utilise un modèle de « copy on write », tant que les éléments de Zygote ne sont pas modifiés par l'application, toutes ces zones mémoires sont partagées par toutes les applications.

La nouvelle instance de Zygote attribue des `id` utilisateurs Linux avant de lancer la classe demandée.

De par la construction évoquée ici, l'ASLR est attribué au hasard lors du lancement de Zygote, mais ensuite toutes les applications Android partagent la même adresse aléatoire. Ainsi, une application peut découvrir les adresses mémoires de toutes les applications. Cela réduit l'impact de cette protection.

Zygote a également été renforcé en 4.3 avec le paramètre `NO_NEW_PRIVS` pour interdire aux applications d'escalader les privilèges lors du lancement d'un autre processus. De même, Zygote supprime les Capability non nécessaires avant d'exécuter les nouvelles instances.

Les scripts d'inits sont également vulnérables aux attaques de « suivit des liens sur les fichiers ». Cela peut permettre d'exploiter un comportement du script pour modifier un fichier sensible. Pour éviter cela, Android lance maintenant les scripts avec le paramètre `O_NOFOLLOW`. Ainsi, les liens ne sont plus suivis.

Néanmoins, des vulnérabilités découvertes sous Linux peuvent parfois s'appliquer à Android. Par exemple, une vulnérabilité dans le noyau sur l'exploitation des futex est parfois exploitable sur certains terminaux.

Une autre vulnérabilité majeure a été découverte dans bash par Stéphane Chazelas, un Français. Mais, coup de chance, Android utilise « Almquist shell bash » (ash) qui n'est pas vulnérable.

Un des moyens pour devenir `root`, consiste à utiliser la partition de flash pour injecter un programme `su`, avec les flags `setuid`/`setgid`. Ces derniers permettent de lancer un programme avec les privilèges du propriétaire du fichier. Si le programme `su` appartient à `root`, alors l'utilisateur peut lancer un shell sous `root`.

Google a supprimé tous les programmes ayant ces flags activés pour réduire la surface d'attaque.

De plus, depuis Android 4.3, la partition `/system` de l'OS interdit l'exploitation de ces flags (`nosuid`). Il n'est plus possible de devenir `root` simplement, avec un fichier lui appartenant.

Pour contourner cela, les développeurs de `su` doivent appliquer une autre approche plus complexe. Le script d'initialisation est modifié pour lancer un service en tâche de fond sous `root`. Le programme `su` demande alors à ce service de lancer un processus fils sous `root`. Pour éviter une exploitation malveillante du `root`, un privilège spécifique est maintenant nécessaire. Une application signale alors qu'elle peut devenir `root`. L'utilisateur peut accorder ce privilège majeur pour une certaine durée ou pour toujours.

Niveau 4 : l'architecture

Pour isoler les applications les unes des autres, Android a détourné le modèle d'utilisateur de Linux. Normalement, chaque utilisateur possède un numéro unique. Ensuite, toutes les applications lancées par l'utilisateur ont un accès intégral à tous les fichiers de ce dernier. Ce modèle est trop permissif.

Android a décidé d'attribuer un utilisateur Linux différent par application. Il y a quelques numéros d'utilisateurs réservés par le système pour certains services. Par exemple, un numéro est réservé pour l'accès au réseau. Aucun périphérique hardware ne peut être accédé par les applications. Les numéros des utilisateurs ne sont pas autorisés. Il faut impérativement passer par le framework pour avoir accès aux coordonnées GPS, à la couche radio, aux vibreurs, etc.

C'est Zygote, présenté ci-dessus, qui se charge d'attribuer l'utilisateur au processus. Un répertoire est également attribué à chaque application. Ce dernier permet à l'utilisateur correspondant d'y avoir un accès total, mais pas aux autres utilisateurs, donc pas aux autres applications. Ainsi, une application X ne peut aller voir les fichiers de l'application Y.

Il existe des paramètres subtils permettant de partager le même numéro d'utilisateur entre plusieurs applications. Ainsi, elles peuvent également partager leurs répertoires de travail. Pour que cela soit possible, il faut que les différentes applications aient été signées par la même signature numérique.

Pour l'accès à la carte SD, Android évolue également. Initialement, la carte était considérée comme ouverte à toutes les applications. Si une application désire utiliser un répertoire à elle pour y stoker des données, elle devait avoir le privilège de lire et d'écrire sur l'intégralité de la

carte. Maintenant, il n'est plus nécessaire d'avoir ces privilèges. Un répertoire sur la carte SD est attribué à chaque application. Elle peut y accéder sans contrainte. Ainsi, elle n'a pas pour autant accès à l'intégralité de la carte SD.

Cette approche permet une isolation forte entre les applications au niveau disque et SE Linux.

Pour les tablettes, Android propose maintenant la possibilité d'avoir plusieurs utilisateurs humains différents. Il a fallu alors ajouter cette notion au-dessus des utilisateurs Linux utilisés pour isoler les applications.

Pour régler cela, Android utiliser un répertoire `/data` par utilisateur. Lorsqu'un utilisateur se connecte, des liens dans le système de fichiers permettent de donner accès aux seuls fichiers de l'utilisateur. Les applications ne savent pas qu'elles utilisent maintenant un répertoire différent par utilisateur.

Si plusieurs utilisateurs humains installent la même application, Android le détecte et n'installe qu'une seule instance dans le téléphone. Il faut que tous les utilisateurs désinstallent l'application pour qu'elle disparaisse réellement du terminal.

Niveau 5 : le framework

Contrairement à d'autres systèmes, Android est un système très ouvert. Les applications peuvent communiquer entre elles en exploitant de nombreux canaux différents. Par exemple, il est possible de se placer en écoute sur un socket dans une application. Une autre application peut ainsi y accéder. (Certains OS mobiles ne permettent pas cela).

Pour communiquer entre les processus, Android propose un module dans le noyau appelé le Binder. C'est un composant essentiel à Android. Il a été conçu pour Palm, il y a des années, avant d'être enrichi pour rejoindre Android. Ce composant utilise la mémoire partagée pour transmettre des messages entre les applications. Les messages peuvent être du type « send and forget » ou attendre une réponse.

Le Binder est un élément très important de la sécurité d'Android. En effet, le module noyau va injecter dans chaque message, l'`UID` (user id) et le `GID` (group id) de l'émetteur. Ainsi, le destinataire du message est en capacité de vérifier les privilèges de l'appelant pour lui autoriser, oui ou non, à exploiter le service. C'est ce mécanisme qui permet de protéger l'accès à tous les composants hardwares. Une seule application est autorisée à y avoir accès. Cette application attend des messages via le Binder pour y répondre. Mais avant, elle vérifie les privilèges de l'appelant. S'il n'a pas les privilèges d'accès au GPS par exemple, il recevra un message d'erreur.

Le Binder est la clé de voûte des privilèges Android. Les composants voulant être invoqués doivent implémenter une interface décrite dans un fichier AIDL.

Tous les objets implémentant l'AIDL peuvent être invoqués par d'autres applications. Le framework de RPC est également capable de communiquer avec un objet sans sortir du processus, sans passer par le module du noyau. Cela permet une communication plus rapide.

Un `ContentProvider` comme tous les objets exploitant le `Binder` peut être exportable (`exported=true`). Cela indique que l'objet AIDL correspondant peut être invoqué depuis une autre application. Avec la valeur `false`, le composant ne peut être invoqué que depuis le même processus. Il est de la charge du développeur d'imposer des privilèges chez l'appelant lors de l'exposition d'un provider aux autres applications.

Niveau 6 : les privilèges

Les privilèges ne sont pas limités à une liste. Les développeurs peuvent en ajouter de nouveau. Un privilège est accordé à une application lors de son installation. Il peut être obtenu, soit après accord de l'utilisateur, soit parce que l'application qui le demande possède la même signature numérique que l'application qui le déclare. Il existe également des privilèges systèmes qui ne peuvent être accordés qu'aux applications signées par le constructeur, ou venant d'applications du répertoire `/system` du terminal. Ce dernier n'est pas

modifiable.

Lorsqu'une application est mise à jour avec de nouveaux privilèges, l'utilisateur en est informé.

Plusieurs applications peuvent déclarer le même « nouveau privilège », avec des contraintes différentes. Si c'est le cas, **la première application installée impose son point de vue. Cela peut permettre de compromettre un privilège limité à la signature par exemple.**

Une application X déclare et demande le privilège P de l'application A pour l'ouvrir à tous. L'application A est installée par la suite, sur les conseils de X. X est alors en capacité d'exploiter les privilèges de A qui pourtant, sont normalement limités aux signataires de A.

Pour s'en protéger, les applications doivent ajouter des tests pour vérifier la consistance du privilège déclaré.

De nouveaux privilèges sont parfois ajoutés par Google. **Si une application déclare un de ces nouveaux privilèges pour une application installée sur un OS ancien, l'application peut en bénéficier silencieusement lors d'une mise à jour de l'OS.**

Comme Android permet une communication forte entre les applications, il existe un risque d'attaque entre les applications installées dans le téléphone. Par exemple, avant, les `ContentProvider` (fournisseurs de contenu) étaient, par défaut, accessibles par les autres applications. Ce n'est plus le cas depuis la version de l'API 17. Comme les providers sont généralement implémentés comme des façades d'une base SQLite, les attaques de type SQL injection sont possibles. Il est important de fermer ces accès ou de les contrôler par un privilège spécifique.

D'autres vulnérabilités ont été découvertes chez certains fournisseurs. Par exemple, des services d'applications privilégiées étaient ouverts. Ils étaient alors possibles de les exploiter pour bénéficier des privilèges systèmes de certains services de Samsung. Cela permettait d'installer silencieusement des applications ayant tous les privilèges.

Depuis la version 4.2, Android, possède une liste noire des numéros de SMS surtaxés pour chaque pays. Avant qu'une application n'envoie un de ces SMS, une confirmation supplémentaire de l'utilisateur est nécessaire.

Niveau 7 : Chiffrement

Il est possible d'imposer un VPN pour l'ensemble du terminal ou pour certaines applications. Depuis Android 4.4, il est également possible d'utiliser des paramètres différents pour chaque utilisateur. Il est possible d'avoir simultanément plusieurs flux. La table de routage peut utiliser des flux non chiffrés pour les jeux, et un VPN pour les applications d'entreprises.

Initialement, le chiffrement sous Android utilisait une implémentation en Java (BouncyCastle). Maintenant, le code est un wrapper vers OpenSSL. Les algorithmes proposés évoluent au fil des versions.

L'algorithme de générateur aléatoire sécurisé a été renforcé, suite à la découverte que l'entropie n'était pas suffisante. En 4.3, l'entropie est également améliorée lors du démarrage et de l'arrêt du terminal, afin d'éviter sa prédiction lors du provisionning de plusieurs terminaux.

Android ne proposait pas de conteneur sécurisé où placer les secrets des applications. De nombreuses étapes ont été nécessaire pour offrir enfin une API correcte. Initialement, un démon codé en C avait pour charge de maintenir les clefs privées nécessaires aux VPN. Ce composant a évolué pour permettre d'y stocker des clés privées pour les applications. L'API permet alors de demander à l'utilisateur de sélectionner une clef privée pour une application donnée. Une base de données garde cette relation. Telle application peut accéder à cette clé privée.

Il est également possible d'avoir accès à un conteneur de clés privées exclusif à l'application. Ainsi, l'utilisateur n'intervient plus. Charge à l'application d'utiliser cette clé pour chiffrer ses données. L'implémentation est solide. Impossible de sortir un clé privée pour l'envoyer à une

autre application. En fait, le code possède un wrapper vers la clé privée maintenue dans le conteneur hardware. Le nécessaire est proposé pour pouvoir utiliser cette clé pour une connexion HTTPS avec authentification mutuelle par exemple.

Si l'application doit pouvoir lire la clé privée, il faut procéder en trois étapes. Générer une clé privée pour l'application. Générer un password. Chiffrer ce password avec la clé privée. Sauver le password chiffré dans un fichier. Générer une autre clé privée, mais en dehors du conteneur hardware. Chiffrer cette clé avec le mot de passe généré.

Ainsi, une clé privée manipulable est disponible, protégée par un mot de passe qui est lui-même protégé par une clé privée hardware.

Pour protéger les clés physiquement, une première approche consistait à utiliser le coffre-fort lié au NFC. Comme tous les téléphones n'en sont pas pourvus, maintenant, les secrets sont gardés par un OS parallèle proposé par les chips ARM. Lors du lancement de l'OS, deux systèmes isolés sont lancés. Quelques trames peuvent être échangées entre les OS. L'OS sécurisé garde le secret majeur protégeant les clés privées. Depuis Android 4.3, les applications peuvent demander à savoir si le certificat est protégé par le Hard.

Pour renforcer la sécurité des applications, des API permettent maintenant d'imposer l'utilisation d'un VPN pour une application.

Certains pays ou entreprises utilisent une autorité de certification valide, pour se placer en homme-du-milieux lors d'une communication chiffrée. Le principe est le suivant. Un serveur chiffré doit présenter un certificat signé par une autorité de confiance. En se plaçant au milieu du flux, l'État déchiffre le flux, et présente un flux chiffré avec un autre certificat signé par une autre autorité valide. C'est comme cela que des mails ont été capturés lors des révolutions d'Afrique du Nord.

Pour éviter cela, il existe un protocole qui permet de mémoriser l'autorité associée à un site. Ensuite, si un certificat valide est présenté, mais avec une mauvaise autorité, le certificat est rejeté.

Depuis Android 4.2, ce mécanisme est mis en place dans les téléphones. Android 4.4 traite spécifiquement les sites de Google.

Android permet de déplacer une application sur la carte SD. Cette dernière pouvant être facilement extraite du téléphone, cela pourrait représenter un risque. Pour éviter cette faile, un secret est généré lors du tout premier boot du terminal. Ce secret permet de chiffrer les applications déplacées sur la carte SD. Ainsi, même placées dans un autre terminal, les applications ne sont pas exploitables.

De même, pour les grosses applications, il existe la technologie APK Extensions File. Il s'agit des fichiers chiffrés d'extension `.obb`.

Niveau 8 : Authentification

Le framework Android permet de proposer un Authentification Provider. Cela permet à des applications comme Facebook, Twitter ou Google de proposer les comptes des utilisateurs, sans pour autant diffuser le mot de passe. C'est une implémentation de OAuth2 à l'intérieur du périphérique. Vous retrouvez les providers dans la lise des comptes des paramètres d'Android.

Niveau 9 : Adb

ADB est un démon qui permet une manipulation de l'OS via un câble USB. Avant Android 4.3, il suffisait de brancher le téléphone à un PC pour qu'un virus puisse communiquer avec le téléphone et installer des applications malveillantes.

Maintenant, il est beaucoup plus difficile de jouer ce scénario. En effet, l'accès au déboguage n'est pas actif par défaut. Pour l'activer, l'utilisateur doit connaître la procédure volontairement complexe. Il doit afficher les numéros de version du téléphone, appuyer une

dizaine de fois sur le numéro du build. Cela ajoute un menu développeur. Il doit enfin activer le déverminage par USB. Depuis 4.3, ce n'est pas terminé. Il doit en plus, accepter de partager une clef entre le PC et le téléphone.

Ainsi, un utilisateur normal n'activera pas cette fonctionnalité. Un développeur avancé qui l'a activée, ne permettra pas à un voleur de se connecter au téléphone par USB, sans débloquer l'écran du téléphone.

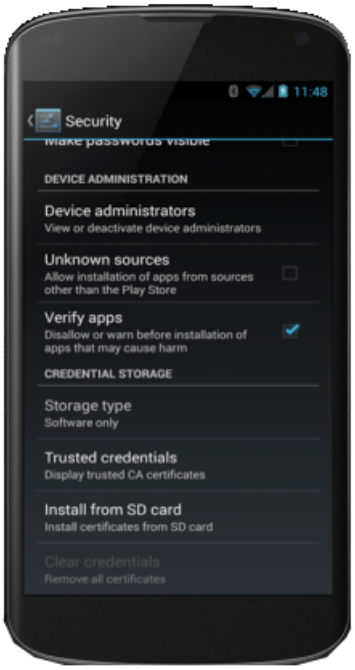
Niveau 10 : Installation depuis le PlayStore

Le PlayStore joue un rôle important dans la sécurité de la plate-forme. Il vérifie les informations de signature du développeur. Ce dernier doit présenter une carte de paiement valide pour pouvoir publier une application. La responsabilité individuelle est liée à la carte de paiement utilisé.

En pratique, avec une carte volée, il est possible de publier une application sous une fausse identité.

Par défaut, seule les applications venant du Play Store peuvent être installées. Cela réduit la flexibilité des attaques (difficulté de faire télécharger et installer un APK depuis un site malveillant par exemple).

Pour accepter les applications ne venant pas d'un store officiel, l'utilisateur doit activer une case à cocher dans un paramètre difficile d'accès. Une alerte l'informe alors du risque qu'il prend. La plupart des utilisateurs n'activent jamais ce paramètre ou font marche arrière lorsque l'alerte est présentée.



Google vérifie autant que possible les applications avant de les diffuser. Il utilise pour cela les bases de données des Anti-virus, tous les nouveaux outils d'audits proposés par les chercheurs (audit des sources, décompilations automatiques, etc). Il lance également l'application dans un Sandbook pour identifier son comportement. Les analyses s'effectuent à plusieurs niveaux : de l'analyse statique du code, à l'analyse heuristique sur des comportements applicatifs, à la signature des développeurs ou l'analyse manuel d'application par les chercheurs.

Après un certain temps, si rien de troublant n'a été identifié, l'application est disponible au téléchargement.

Il y a eu des applications malveillantes qui ont réussi à contourner cela. Certains se sont même amusés à exploiter l'exécution effectuée par Google dans la sand-box

pour en prendre la main et l'analyser.

Cela n'est pas plus dangereux qu'un antivirus classique qui est toujours en retard d'une attaque. Aucune solution, automatique ou manuelle, ne peut garantir une sécurité absolue.

En cas de découverte tardive d'une application malveillante, Google la désinstalle automatiquement sur tous les téléphones et le développeur est banni à vie, sur son compte et tous les comptes liés un jour ou l'autre, de tous les services de Google.

Certains ont eu la désagréable surprise de se faire bannir pour avoir publié une application utilisant les vidéos Youtube. L'auteur ainsi que sa famille ne peuvent plus utiliser les périphériques Android.

Ayant les privilèges systèmes, le Play Store peut appliquer des modifications profondes dans l'OS. Des chercheurs proposent d'intervenir dynamiquement en mémoire pour patcher les failles du SDK à chaud dans les applications. Ainsi, il n'est plus nécessaire d'attendre une hypothétique mise à jour de l'OS par les constructeurs pour bénéficier des corrections. Il semble qu'une prochaine version du Play Store soit capable de faire cela. Le framework Xposed (nécessitant `root`) propose une approche similaire. Il permet d'intervenir uniquement en RAM pour modifier l'OS ou les applications.

Niveau 11 : Vérification de toutes les applications

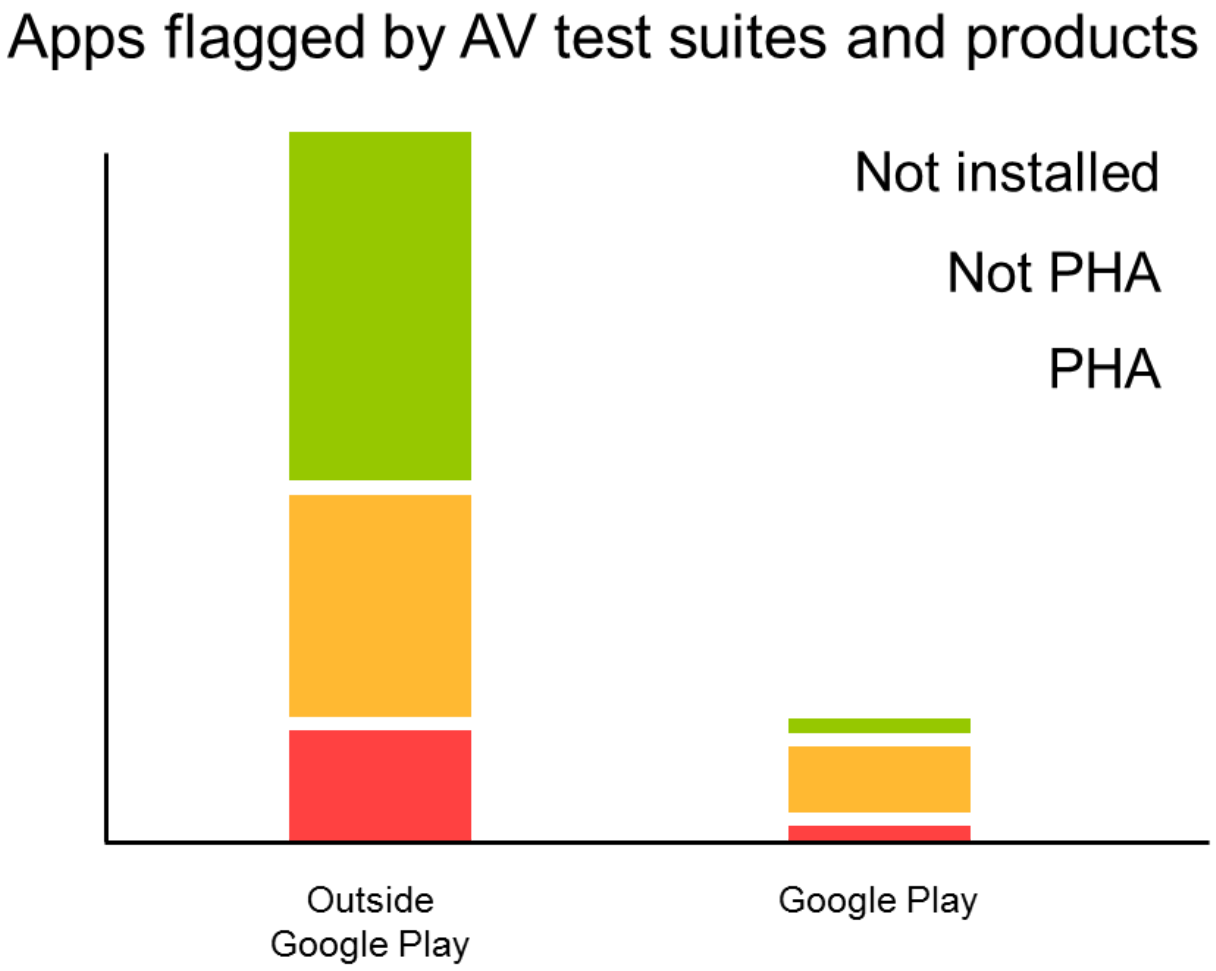
Depuis Android 4.2, un paramètre activé par défaut permet de demander à Google Play de vérifier les applications avant leurs installations, même si elles ne viennent pas du Play Store ou d'un autre store préinstallé. C'est une sorte d'Anti-virus géré par Google. Par jour, plus de dix millions d'applications sont ainsi vérifiées. Une application inconnue est envoyée chez Google pour analyse ultérieure. Cela permet de renforcer la sécurité dans le temps.

En effet, la plupart des applications malveillantes viennent d'applications ayant été installées en dehors du Play Store. Certains pays n'ont en effet par accès à la place de marché de Google. Certains utilisateurs ne souhaitent pas payer les quelques euros pour récompenser le travail des développeurs et préfèrent utiliser des applications pirates, avec tous les risques que cela comporte.

Si une application est identifiée comme malveillante, un message d'alerte est envoyé à l'utilisateur et l'application n'est pas installée. Il doit être particulièrement motivé pour imposer l’installation en désactivant la vérification des applications avant leurs installations.

Google ajoute des millions d'informations complémentaires par jour pour renforcer la sécurité. Il exploite tout ce dont il a accès pour identifier les menaces au plus vite.

Une analyse comparée avec les Anti-virus montre que la plupart des signatures des antivirus concernent des applications que Google ne rencontre jamais dans les téléphones. Ou bien, les antivirus alertent sur des applications n'étant pas malveillants.



Niveau 12 : Verrou de l'écran

L'utilisateur peut ajouter un verrou pour bloquer l'accès au téléphone. Ce verrou peut être un schéma, une suite de chiffres ou un mot de passe. L'utilisateur peut également refuser d'utiliser un verrou. C'est mal.

Certains se sont amusés à découvrir les schémas d'un téléphone, simplement en regardant les traces de doigts sur l'écran. D'autres ont réalisé des robots pour brute-forcer un schéma.

Si le téléphone possède des secrets importants comme une clef privée, l'utilisateur doit obligatoirement posséder un verrou sur l'écran.

Pour éviter les inconvénients de la saisie régulière du code d'accès, Google propose d'utiliser la proximité avec un autre périphérique déverrouillé. Par exemple, une montre Google Wear peut signaler la proximité de l'utilisateur. Ainsi, le schéma n'est pas nécessaire. Mais si le téléphone est oublié ou volé, le schéma doit être saisi.

Attaques sérieuses

Malgré toutes les protections mises en place, il existe des vulnérabilités sérieuses qui affectent Android.

Master-Key (CVE-2013-4787)

Les applications Android doivent être signées pour être installées sur l'OS. Le certificat utilisé

peut être auto-signé ou signé par une autorité. Android possède en dur dans le source, des certificats pour certains développeurs. Ils peuvent ainsi créer des applications avec des privilèges spéciaux. L'un de ces certificats concerne Adobe. Une application signée par celui-ci ou par un certificat issu de celui-ci peut injecter du code dans d'autres applications. Ce privilège a été accordé pour le plugin Flash.

Normalement, un certificat est valide si toute la chaîne de certification l'est. Android n'applique pas une validation conforme. Ainsi, un certificat peut être considéré comme valide s'il se déclare certifié par un des certificats présents dans l'OS.

Cette attaque ne peut s'appliquer qu'aux applications utilisant un composant WebView, pour les versions d'Android inférieur à KitKat.

Google détecte à présent les certificats louches dans les applications du PlayStore.

Heartbleed (CVE-2014-0160)

Une attaque importante de la librairie OpenSSL permet à un attaquant d'avoir accès la mémoire du client ou du serveur d'une communication chiffrée. Les clés privées en RAM ou les mots de passe ne sont alors plus sécurisés.

Depuis la version 4, Android utilise OpenSSL. Les versions 4.1.1 sont vulnérables à cette faille (vidéo goo.gl/S0dhAF).

Privacy Disaster (CVE-2014-6041)

Une faille importante a été découverte dans l'implémentation du navigateur par défaut d'Android. En injectant un caractère zéro dans une URL Javascript, il est possible de contourner les protections d'isolations entre les sites (SOP – Single Origin Policy).

```
<iframe name="test" src="http://www.rhainfosec.com"></iframe>
<input type=button value="test"
onclick="window.open('\u0000javascript:alert(document.domain)','test')" >
```

Ainsi, il est possible de voler le cookie d'un site A, en visitant un site B. Utilisez Chrome sous Android et désactivé le navigateur par défaut pour vous en prémunir.

addJavascriptInterface (CVE-2012-6636)

En utilisant un composant WebKit et une classe pour lier le code Javascript au code Java, il était possible, avant Android 4.2, de faire exécuter du code java arbitraire depuis une page web. Maintenant, les classes doivent posséder l'annotation `JavascriptInterface`. Gérer la sécurité pour les versions précédentes d'Android n'est pas facile.

UI State Inference Attack

Des chercheurs ont trouvé une technique permettant à une application en tâche de fond de deviner l'application en façade et à quel moment elle change d'activité. Pour cela, ils analysent en permanence des indicateurs de l'OS comme la consommation de la mémoire partagée, l'évolution de la consommation réseau, etc. Après une phase d'apprentissage, l'algorithme est capable de savoir quand Gmail ou une autre application est lancée, et quand une nouvelle activité est ouverte. L'application en tâche de fond lance alors au bon moment une activité en front, simulant l'application réelle. Cette activité peut demander à l'utilisateur son mot de passe par exemple. En toute confiance, il pense le donner à Gmail et non à l'application malveillante.

Une vidéo très convaincante est proposée : http://goo.gl/eKdc8n

J'ai moi-même exploité un paramètre subtil d'Android, l'affinité de thread, pour démarrer une activité par-dessus une application légitime. Cette dernière doit avoir son processus vivant lors de l'attaque. L'idée est de forcer l'activité à se superposer à une activité d'une autre

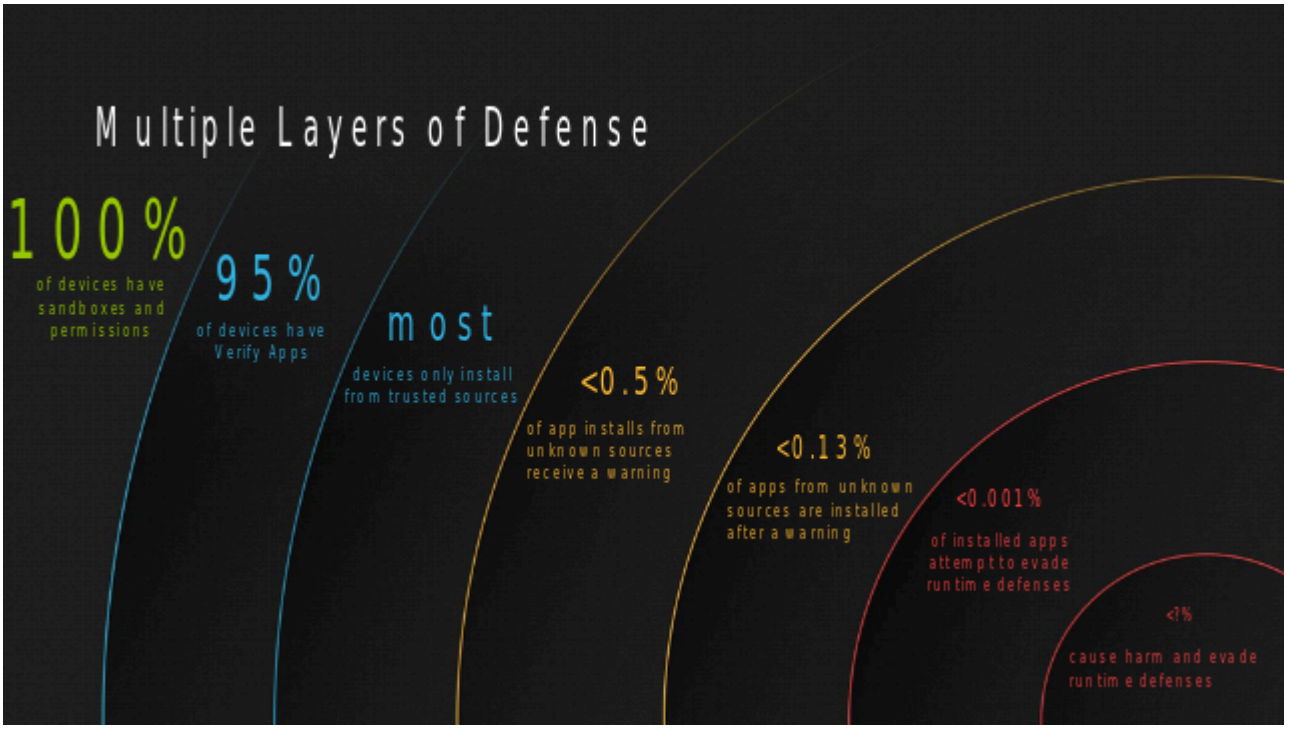
application. Ainsi, je peux afficher une boîte de dialogue sur une activité transparente, pour demander un mot de passe. Derrière la boîte de dialogue, l'utilisateur voit l'application légitime (un mail, la liste des mails, etc.) et saisit naturellement son mot de passe.

Privilèges cachés

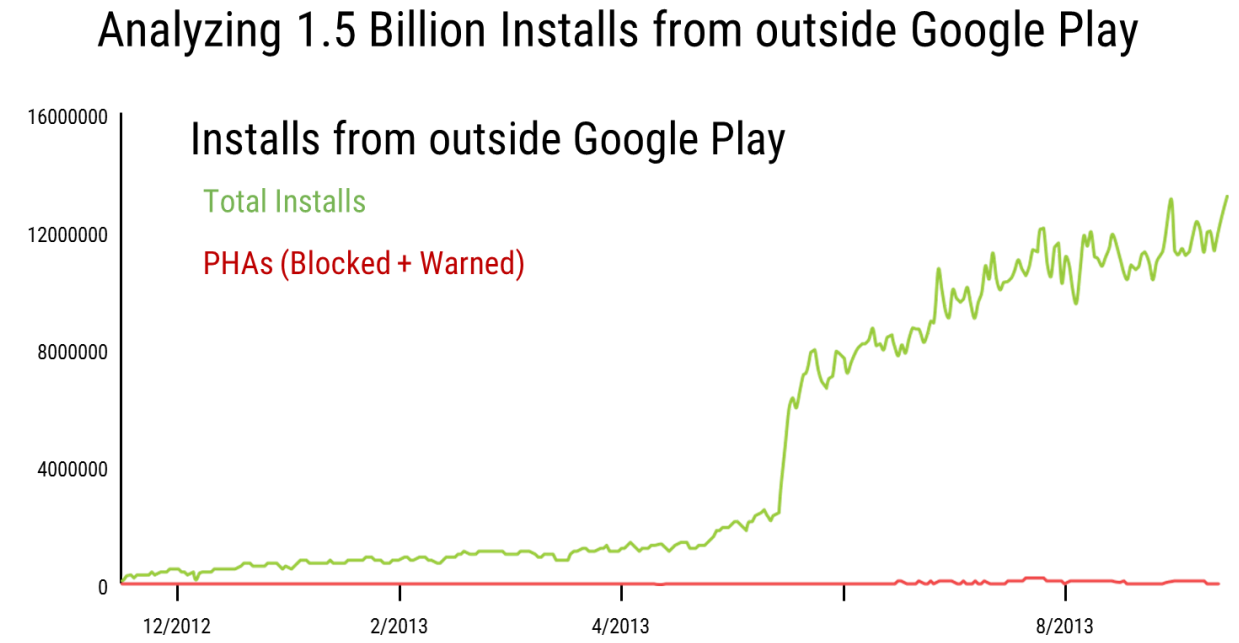
Comme je l'ai déjà expliqué dans un article précédent, des applications venant du même développeur peuvent mélanger leurs privilèges. L'utilisateur accorde un droit pour une application, puis pour une autre. En fait, les deux applications possèdent les privilèges des deux. Sur le Play Store, vous trouverez l'application « Hidden permissions » qui peut révéler les privilèges cachés des applications. L'application « Add Permissions Demo » montre comment une application peut ajouter dynamiquement le privilège SMS. Comparez les privilèges affichés par Google Play avec les privilèges présents dans le menu application du téléphone. Cela ne match pas toujours.

Dans la vraie vie ?

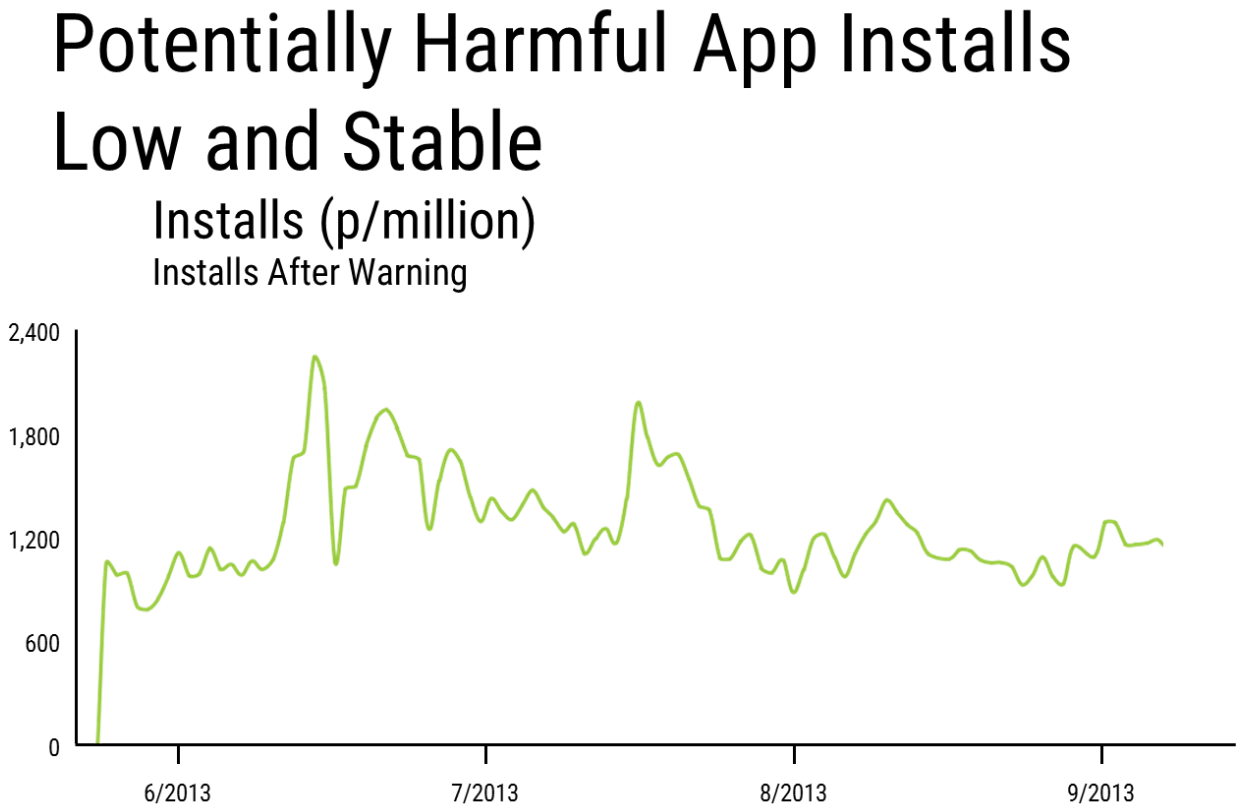
Dans une présentation de 2014, Adrian Ludwig, « Google's Android security chief », est revenu sur les risques réels des terminaux sous Android, vu par Google. Voici les différents niveaux de sécurité.



Google a analysé 1,5 milliard d'applications installées en dehors du Play Store. Très peu ont générée une alerte.



À peu près 1200/millions ont été installé après l'alerte.

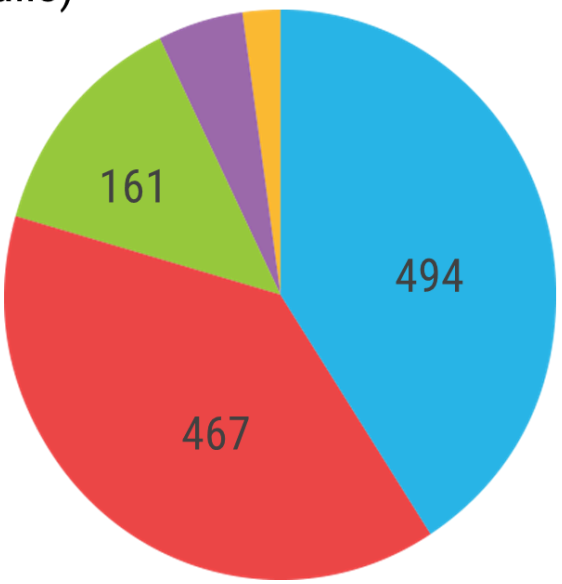


De ces applications, 700/millions présentent un risque (Fraude SMS, Spyware, Trojan, Backdoor, etc.)

Users More Likely to Install Non-Malicious Rooting, and SMS Fraud Apps

PHAs Installed (per million installs)

494	Non-Malicious Rooting
467	SMS Fraud
161	Spyware
54	Generic
20	Trojan
6	Other (Backdoor, Phishing, Malicious Exploit)



Le risque réel pour l'utilisateur est estimé à moins d'une dizaine par million.

Understanding Real Risk to Users

Threat	News Headline	Unique APKs	Installs from Unknown Sources	Installs* (estimated)
Master Key	99% of devices vulnerable	1231	< 8 in a million	<1 in a million (est.)
Droid Kung Fu	Android Malware Steals Sensitive Data Avoids AV Detection	10,000+	<5 in a million	<1 in a million (est.)
Obad	Now being distributed by mobile botnets	14	<1 in a million	<1 in a million (est.)
Androrat	Androrat Remote Access Trojan is Cheaper and More Dangerous than Ever	241	< 1 in a million	<1 in 10 million (est.)

*The total number of installs outside Google Play is not known. This is a blended average across all install channels, assumes that Verify Apps is seeing approximately 50% of installs outside of Google Play.

Les différents niveaux de sécurités permettent de réduire considérablement les risques. Moins de 0.001 % des applications arrivent à s'évader des défenses du Runtime.

Les problèmes majeurs d'Android

Malgré tous ces efforts, Android présente des faiblesses chez les développeurs et les constructeurs.

Les développeurs

Malheureusement, les développeurs d'applications sont fainéants. Ils ne font généralement aucun effort pour sécuriser leurs applications.

Les ressources et autres classes compilées sont accessibles par toutes les applications. Il n'est pas question d'y cacher quelque chose. Pourtant, ils ont une forte tendance à cacher les secrets dans le code source de l'APK ou dans les fichiers de l'application. Une étude sur la grande majorité des applications gratuites du Play Store a montré que les tokens des API ou des comptes Amazon sont facilement accessibles dans les applications. Certains comptes ont été fermés chez Amazon suite à cette découverte. Les développeurs ont dû, en urgence, coder correctement pour pouvoir ouvrir à nouveau le service.

Différents audits de masse ont permit de révéler que les certificats numériques ne sont pas correctement vérifiés. Il est alors facile de déchiffrer les flux d'une application, en se plaçant en « homme du milieu ». Google commence à sévir. Il a envoyé un message aux développeurs pour leur demander de corriger rapidement cela. Sinon, leurs applications seront considérées comme risquées.

Les privilèges des composants logiciels comme les activités, les providers ou autres sont rarement indiqués, laissant la place à des attaques entre les applications.

Les auditeurs d'applications mobiles oublient généralement ce type de scénario. Ils vérifient les flux réseaux, le code, mais pas la capacité d'une autre application d'exploiter l'application auditée.

Les constructeurs

Comme nous l'avons vu, Android renforce régulièrement sa sécurité. Soit directement dans l'OS, soit via le Play Store. Contrairement à iOS, les constructeurs ne font aucun effort pour proposer des correctifs sur les téléphones déjà en circulation. C'est une des difficultés majeures de l'OS.

Google impose aux fournisseurs de se tenir à niveau, en renforçant les règles de validation des plates-formes Android des constructeurs. Ainsi, les prochaines versions des OS sont normalement correctement sécurisées.

Les surcouches des constructeurs, en plus de consommer beaucoup de ressources, présentent souvent des vulnérabilités. Comme elles ont généralement un privilège maximum, une vulnérabilité peut permettre l'installation silencieuse d'application ayant tous les privilèges.

Comme nous l'avons évoqué, il existe des techniques pour renforcer la sécurité à chaud, via un patch dynamique de la Dalvik ou des librairies partagées. Espérons que Google prenne ainsi en charge les faiblesses des constructeurs. La plupart des faiblesses peuvent se corriger ainsi, mais pas toutes.

Pour conclure, Android est un système bien plus sécurisé que Windows. Les risques réels sont extrêmement réduits, malgré le bruit médiatique des fournisseurs d'antivirus. Le play store sert d'Anti-virus avant l'installation d'application. Si vous n'abaissez pas la sécurité vous même, n'ayez aucune crainte.

article@prados.fr
OCTO Technology