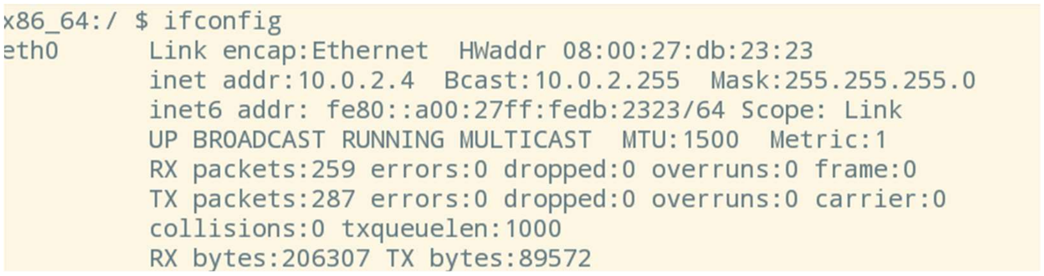
NUID: 001099849

LAB 7: MOBILE SECURITY

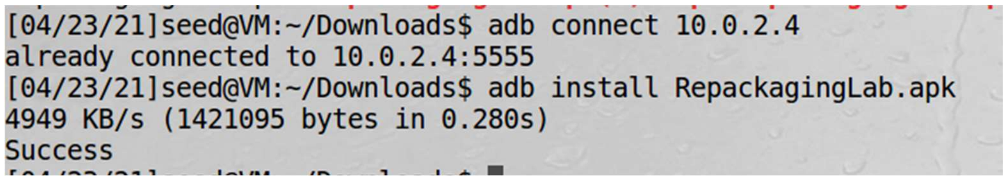
TASK 1: Obtain android apk and install it

IP address of the android 10.0.2.4 as shown in Figure 1. We connected to the android VM using adb and installed RepackagingLab.apk as shown in Figure 2. To verify that we are connected to the VM we can ping using its IP address (Figure 4).



```
x86_64:/ $ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:db:23:23
          inet addr:10.0.2.4  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fedb:2323/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:259 errors:0 dropped:0 overruns:0 frame:0
          TX packets:287 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:206307 TX bytes:89572
```

Figure 1



```
[04/23/21]seed@VM:~/Downloads$ adb connect 10.0.2.4
already connected to 10.0.2.4:5555
[04/23/21]seed@VM:~/Downloads$ adb install RepackagingLab.apk
4949 KB/s (1421095 bytes in 0.280s)
Success
```

Figure 2

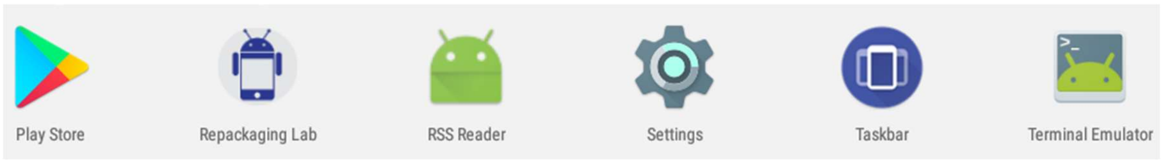
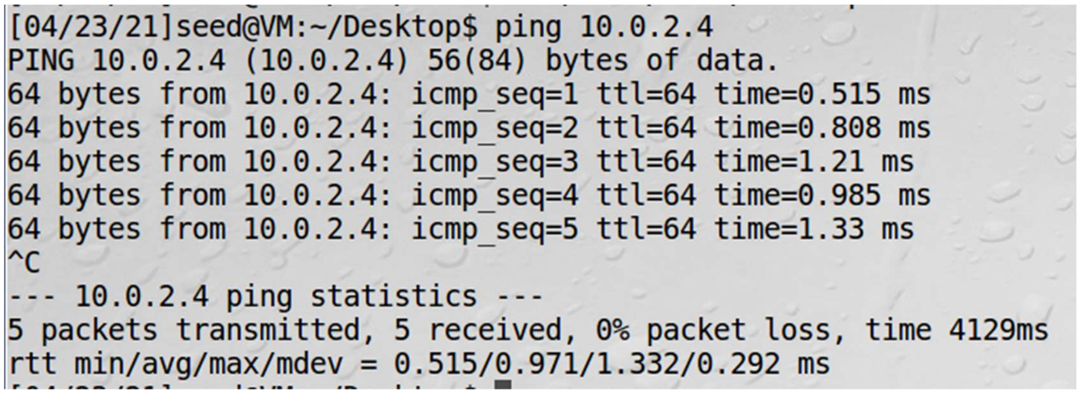


Figure 3

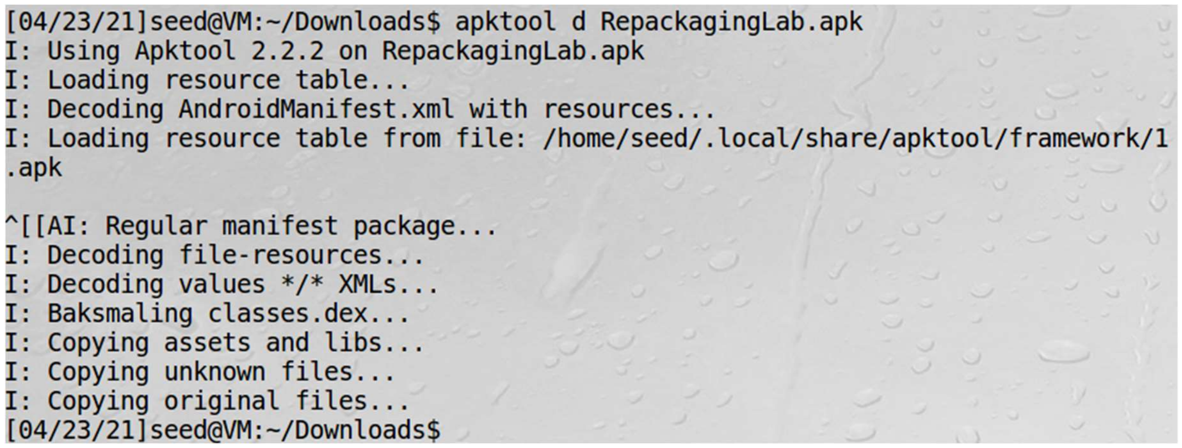


```
[04/23/21]seed@VM:~/Desktop$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.515 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.808 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=1.21 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=0.985 ms
64 bytes from 10.0.2.4: icmp_seq=5 ttl=64 time=1.33 ms
^C
--- 10.0.2.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4129ms
rtt min/avg/max/mdev = 0.515/0.971/1.332/0.292 ms
```

Figure 4

TASK 2: Disassemble the android app

We then disassemble the apk from dex code to more readable format Smali code using apktool with d option as seen in Figure 5.



```
[04/23/21]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

^[[AI: 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...
[04/23/21]seed@VM:~/Downloads$
```

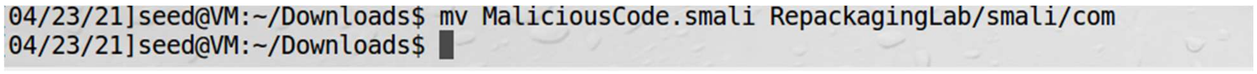
Figure 5

TASK 3: Inject malicious code

The malicious code introduces a new component: Broadcast receiver which is triggered by changing the time of the device (sends time set broadcast).

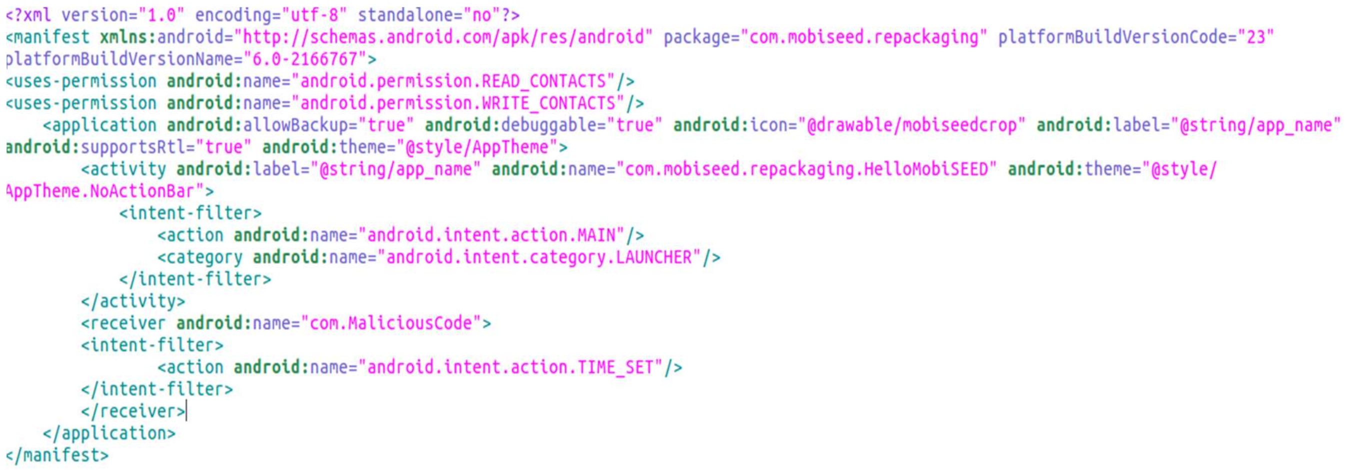
We use ContentResolver object in our application context to access data by querying ContentProvider (which presents stored data as tables to external applications). The Content Resolver performs basic CRUD operations. This in-turn loads CursorLoader to execute this query (get contacts table similar to FROM <table> of SQL) and returns a cursor object as a result. We use this cursor to traverse each contact and delete it.

We installed this malicious smali code from the provided link and moved that file to RepackagingLab/smali/com folder and then edited the AndroidManifest.xml to set required read and write permissions of contacts and register the receiver to TIME_SET event as seen in Figure 7.



```
04/23/21]seed@VM:~/Downloads$ mv MaliciousCode.smali RepackagingLab/smali/com
04/23/21]seed@VM:~/Downloads$ 
```

Figure 6

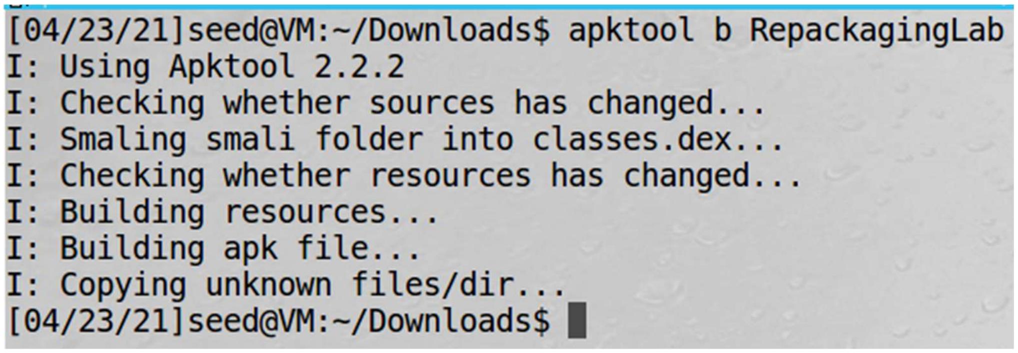


```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.mobiseed.repackaging" platformBuildVersionCode="23"
platformBuildVersionName="6.0-2166767">
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<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:supportsRtl="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>
```

Figure 7

TASK 4: Repackage Android App with malicious code

1) We rebuild the application with the malicious code inserted using apktool with b option as seen in Figure 8.

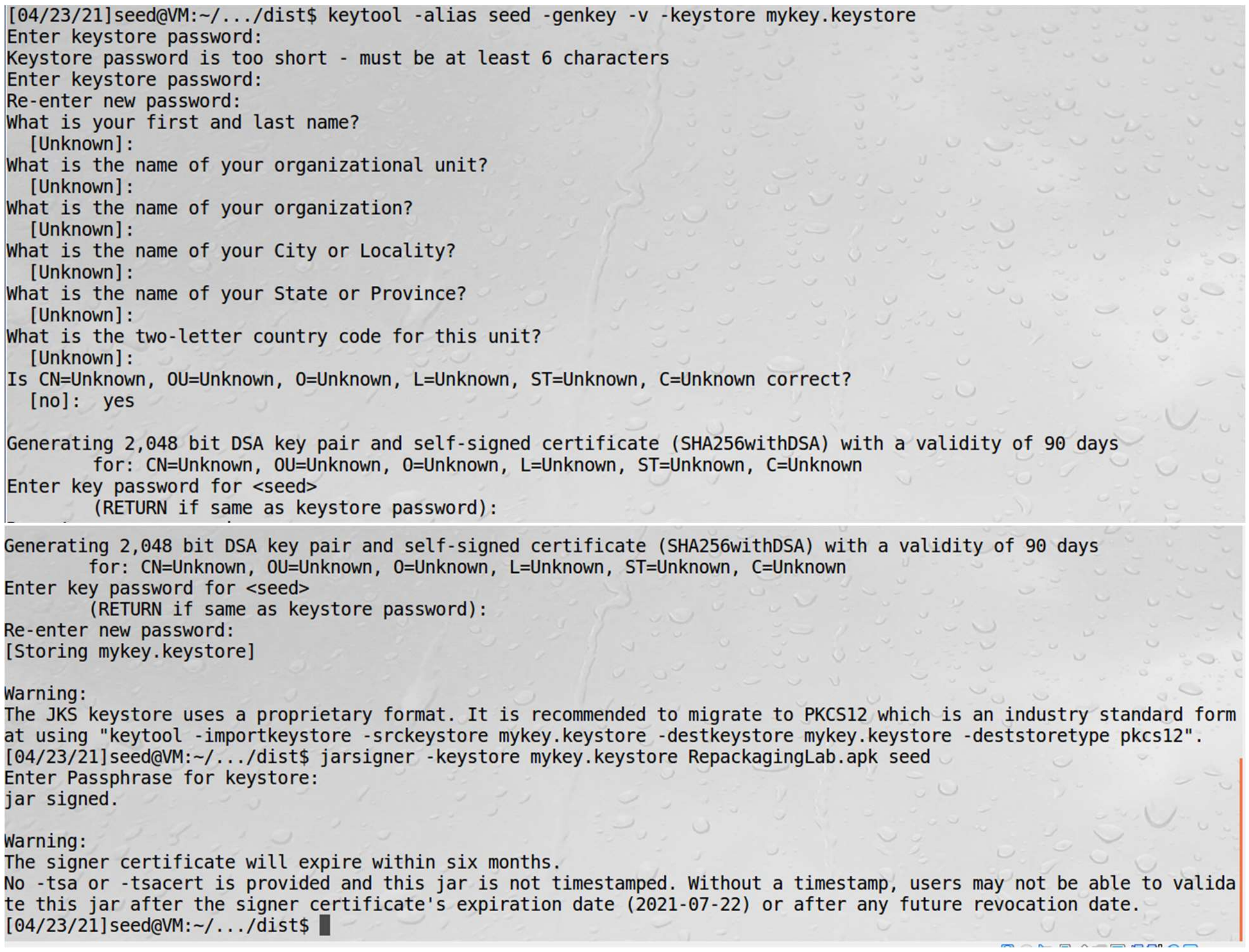


```
[04/23/21]seed@VM:~/Downloads$ apktool b RepackagingLab
I: Using Apktool 2.2.2
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
[04/23/21]seed@VM:~/Downloads$
```

Figure 8

2) Since every android application needs to be digitally signed before it can be installed, we self-sign it. We generate a key pair using keytool command as shown below. The alias name provided is seed and password is then set to access the keystore. Using this we signed the RepackagingLab application.

Although using self-signed certificate is free, from developer point of view, the application store's/device's Trust Store may not have the name using for self-signing hence may block the installation of the app. Also, if name verification is not done by the app stores, any 3rd party with malicious intent can perform the android repackaging attack (like we did in this lab) and sign it using their own name/private key.



```
[04/23/21]seed@VM:~/.../dist$ keytool -alias seed -genkey -v -keystore mykey.keystore
Enter keystore password:
Keystore password is too short - must be at least 6 characters
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
Enter key password for <seed>
        (RETURN if same as keystore password):
```

```
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
Enter key password for <seed>
        (RETURN if same as keystore password):
Re-enter new password:
[Storing mykey.keystore]

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard form
at using "keytool -importkeystore -srckeystore mykey.keystore -destkeystore mykey.keystore -deststoretype pkcs12".
[04/23/21]seed@VM:~/.../dist$ jarsigner -keystore mykey.keystore RepackagingLab.apk seed
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 valida
te this jar after the signer certificate's expiration date (2021-07-22) or after any future revocation date.
[04/23/21]seed@VM:~/.../dist$
```

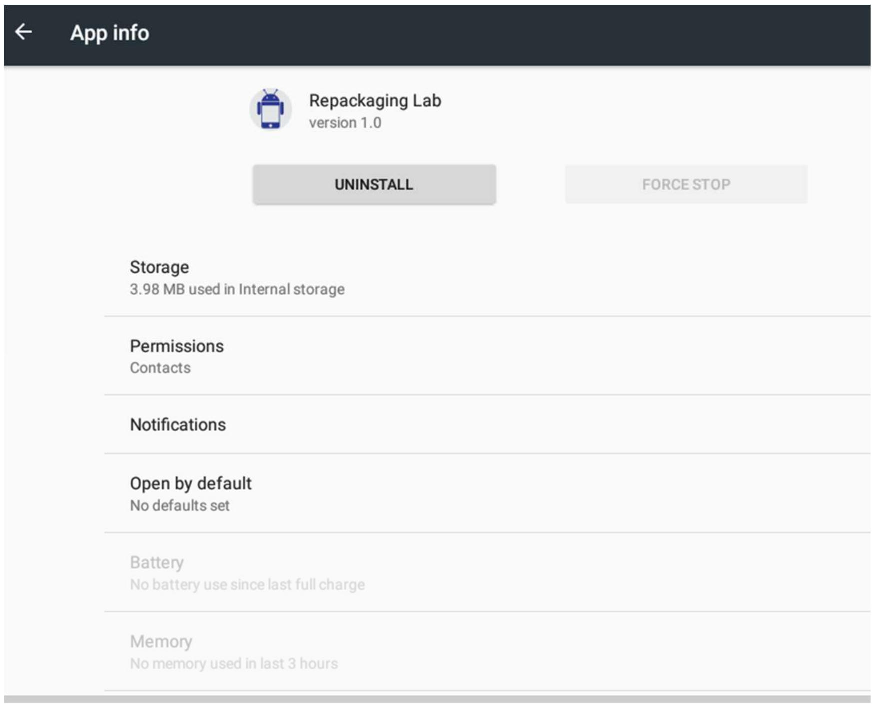
Figure 9: Creating keypair and signing the app

TASK 5: Install the repackaged app and Trigger the malicious code

Since the application was previously installed in the first task, we uninstall it and reinstall the newly built app (Similar to task 1). Once this application is installed, we change the app permission to have access to Contacts as seen in Figure 10. We also add some contacts. When we launch the app, we get the display as seen in Figure 12. To trigger the malicious code, we set the time (here time is set to 12 PM). This event triggers Time_set broadcast which in turn invokes our Receiver and we can see that all the saved contacts are erased successfully.

Countermeasures Against Android Repackaging Attack:

- Code (source code or binary code) Obfuscation makes the attacker difficult to reverse engineer the code to understand and manipulate it.
- Multiple-signature scheme can be used for app signing in place of just self-signing certificates. For example, requiring app market to sign the app in addition to the app pubisher.
- App markets can have name checking in place to ensure that it belongs to the actual legal owner and the name does not exist in the market already.



App info

Repackaging Lab
version 1.0

UNINSTALL FORCE STOP

Storage
3.98 MB used in Internal storage

Permissions
Contacts

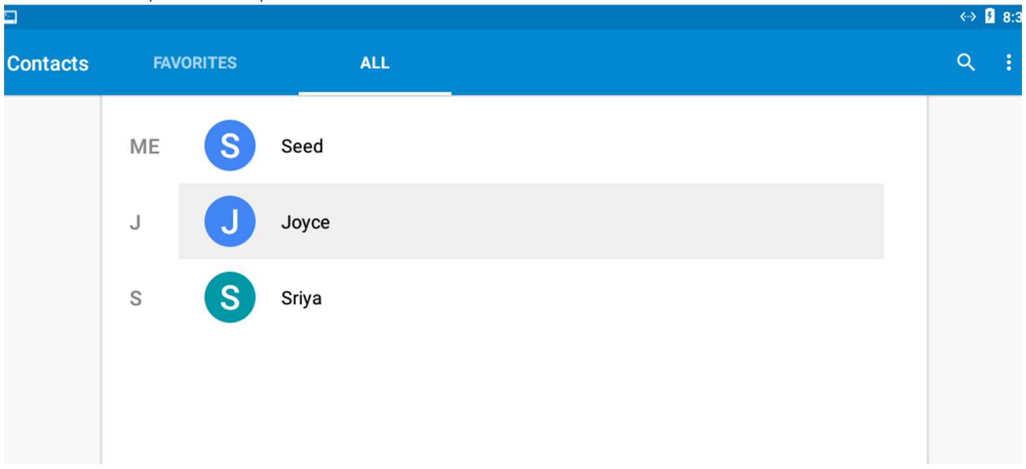
Notifications

Open by default
No defaults set

Battery
No battery use since last full charge

Memory
No memory used in last 3 hours

Figure 10



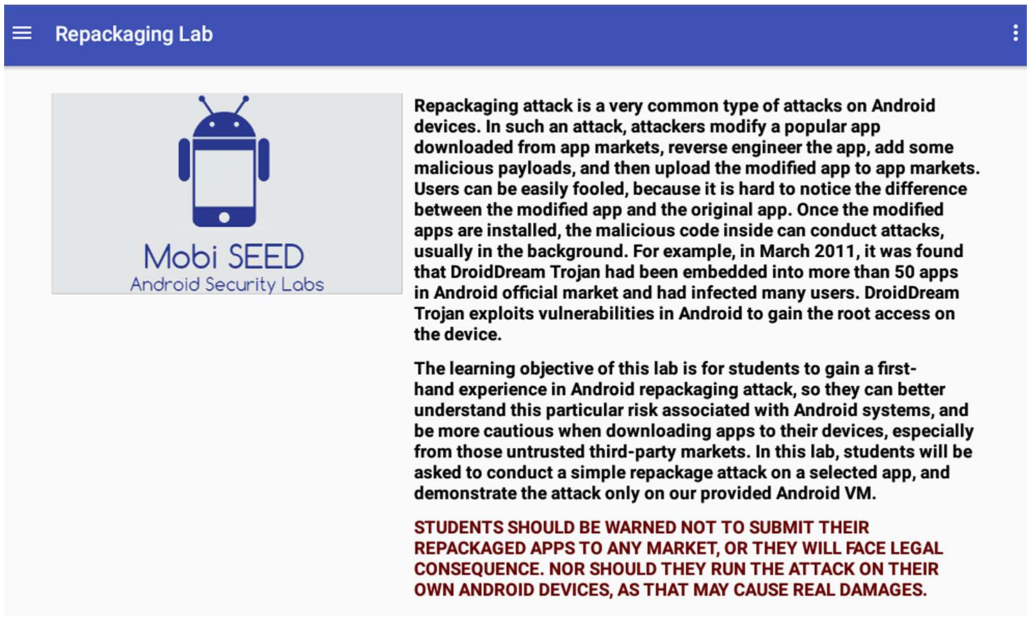
Contacts FAVORITES ALL

ME Seed

J Joyce

S Sriya

Figure 11

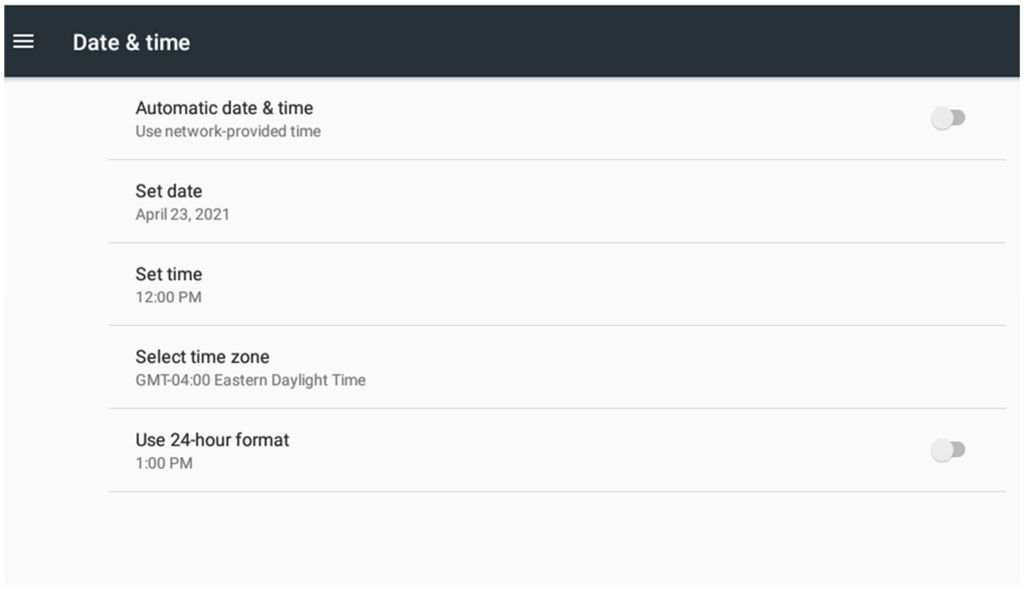


Repackaging attack is a very common type of attacks on Android devices. In such an attack, attackers modify a popular app downloaded from app markets, reverse engineer the app, add some malicious payloads, and then upload the modified app to app markets. Users can be easily fooled, because it is hard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

Figure 12



Automatic date & time
Use network-provided time

Set date
April 23, 2021

Set time
12:00 PM

Select time zone
GMT-04:00 Eastern Daylight Time

Use 24-hour format
1:00 PM

Figure 13

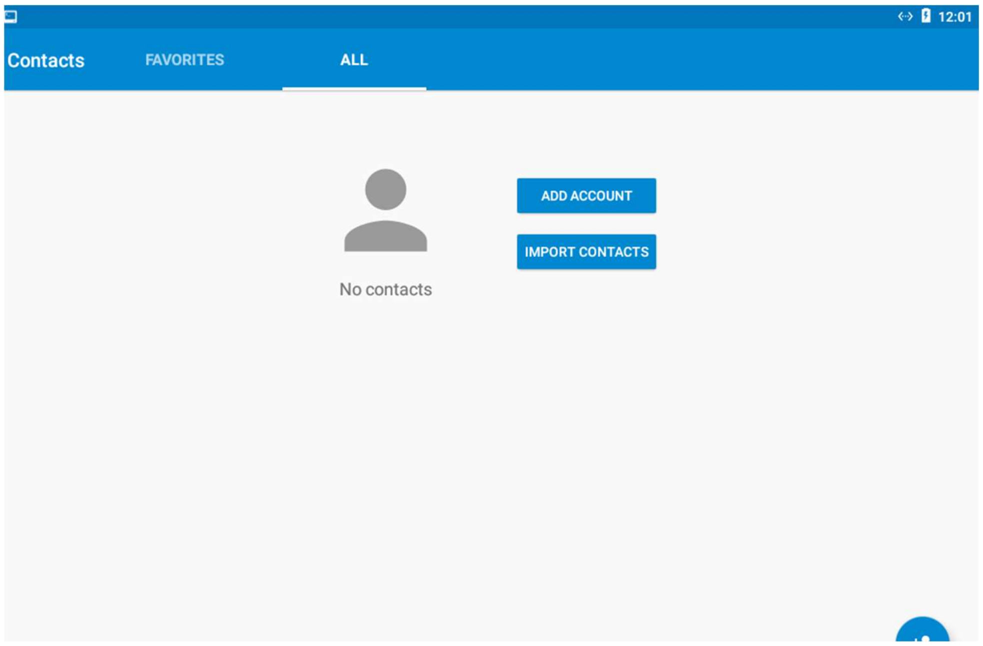


Figure 14

TASK 6: Using repackaging attack to track Victim's location

Step 1: Setting up mock locations

We are using an application with mock locations instead of actual GPS. This app is already installed in the android VM and has 6 locations as seen below.

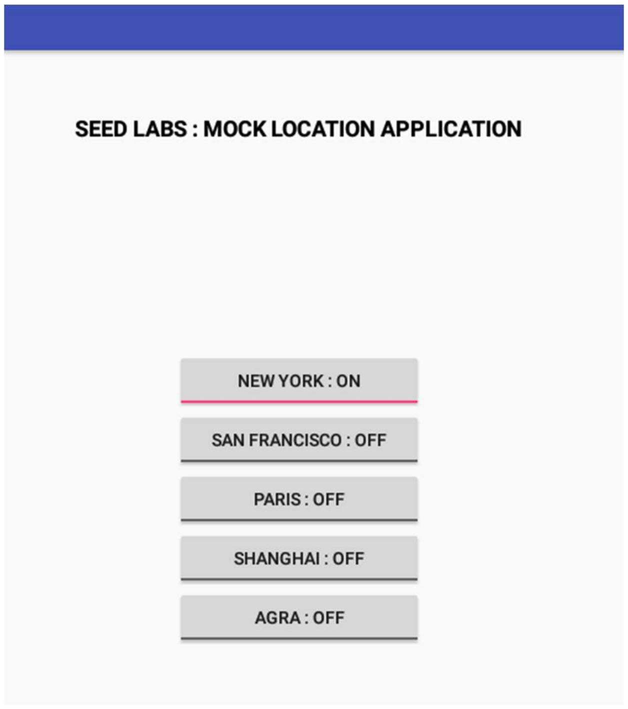
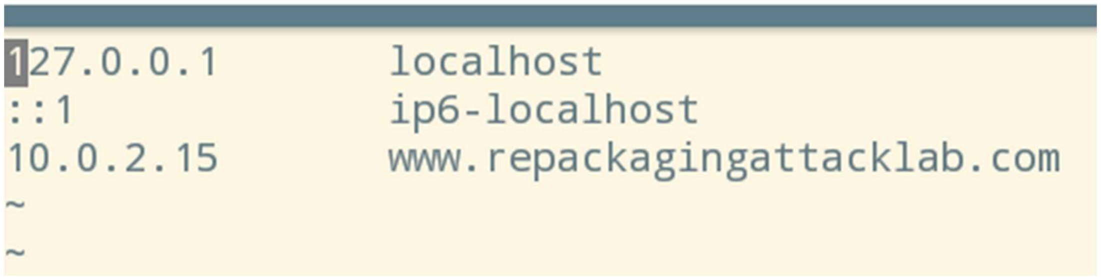


Figure 15

Step 2: Configuring DNS

We send the victim's coordinates to attacker server which will be hosted by our Ubuntu VM. Hence, we will configure Android VM's DNS to hold the IP address of the Ubuntu VM (10.0.2.15)



```
127.0.0.1       localhost
::1             ip6-localhost
10.0.2.15       www.repackagingattacklab.com
~
~
```

Figure 16

Step 3: Repackaging and installing the victim app.

We copy the malicious smali codes provided into the folder RepackagingLab/smali/com/mobiseed/repackaging and then change the AndroidManifest.xml file to set permissions to access locations and internet. We also register this receiver with Time_set event same as the previous attack.

We repackage this app again and self-sign it. Before we install this onto the android, we need to uninstall the existed application and then install this.

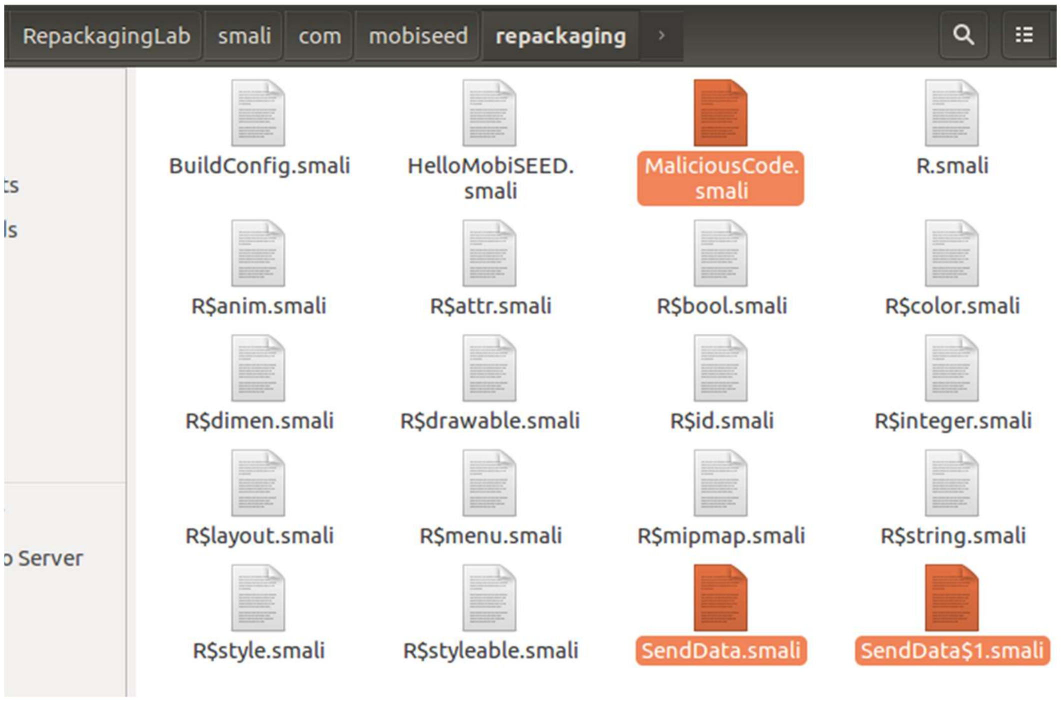
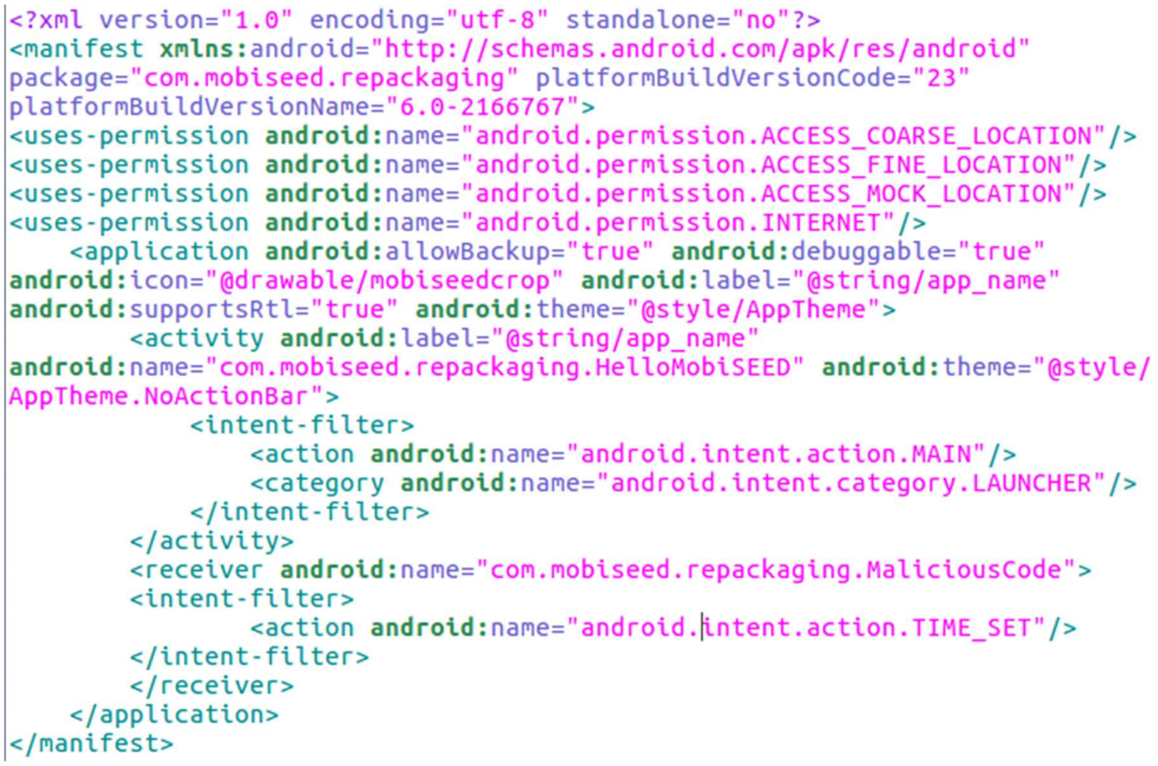
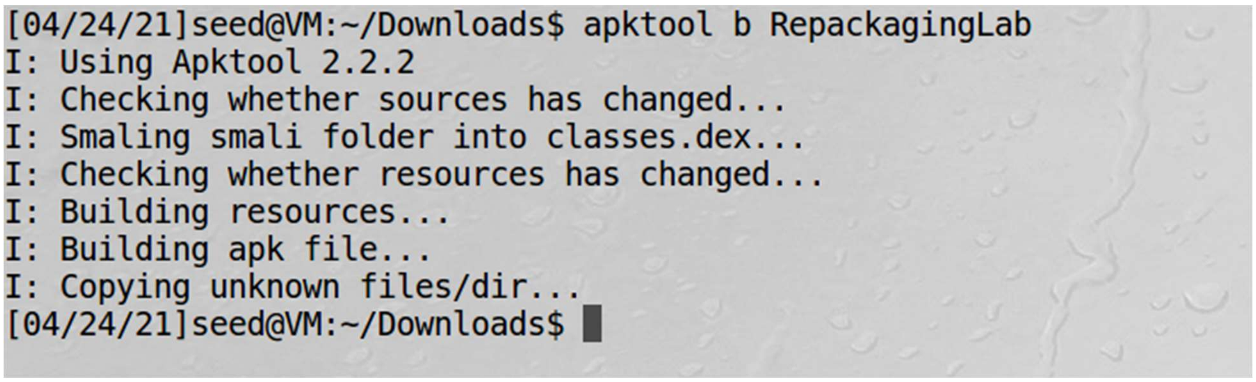


Figure 17



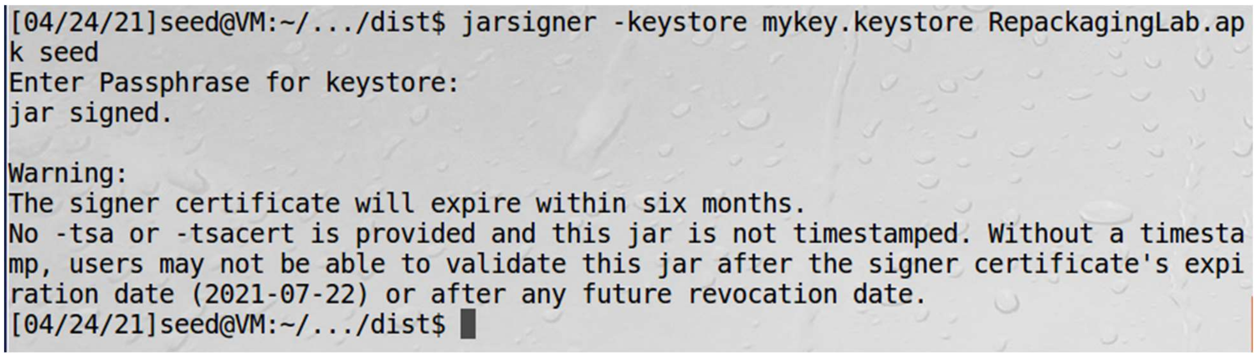
```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.mobiseed.repackaging" platformBuildVersionCode="23"
platformBuildVersionName="6.0-2166767">
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<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/app_name"
android:supportsRtl="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.mobiseed.repackaging.MaliciousCode">
        <intent-filter>
                <action android:name="android.intent.action.TIME_SET"/>
        </intent-filter>
        </receiver>
    </application>
</manifest>
```

Figure 18



```
[04/24/21]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...
[04/24/21]seed@VM:~/Downloads$ 
```

Figure 19



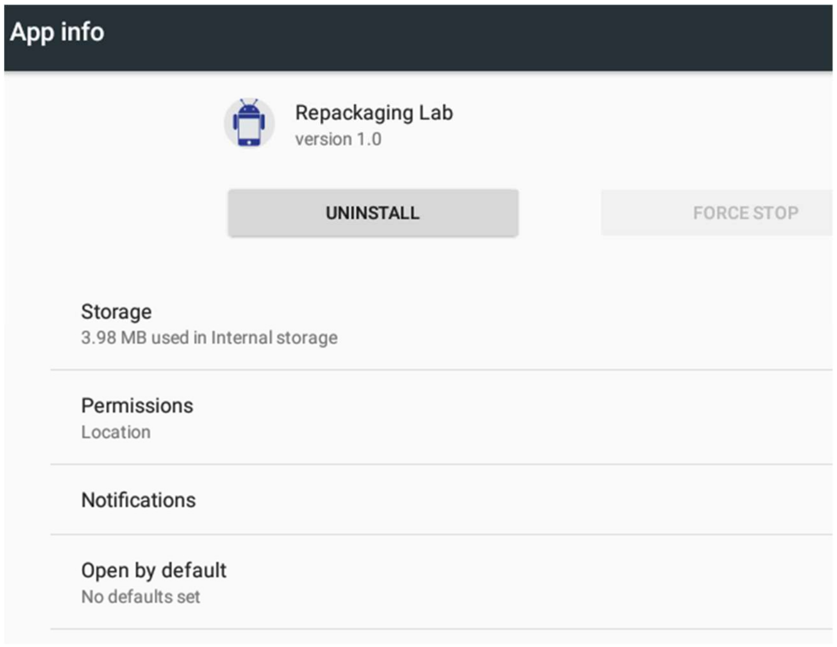
```
[04/24/21]seed@VM:~/.../dist$ jarsigner -keystore mykey.keystore RepackagingLab.ap
k seed
Enter Passphrase for keystore:
jar signed.

Warning:
The signer certificate will expire within six months.
No -tsa or -tsacert is provided and this jar is not timestamped. Without a timesta
mp, users may not be able to validate this jar after the signer certificate's expi
ration date (2021-07-22) or after any future revocation date.
[04/24/21]seed@VM:~/.../dist$ 
```

Figure 20

Step 4: Enabling permissions on Android VM

We change the permissions of the installed RepackaginLab to enable Location permission.



App info

Repackaging Lab
version 1.0

Storage
3.98 MB used in Internal storage

Permissions
Location

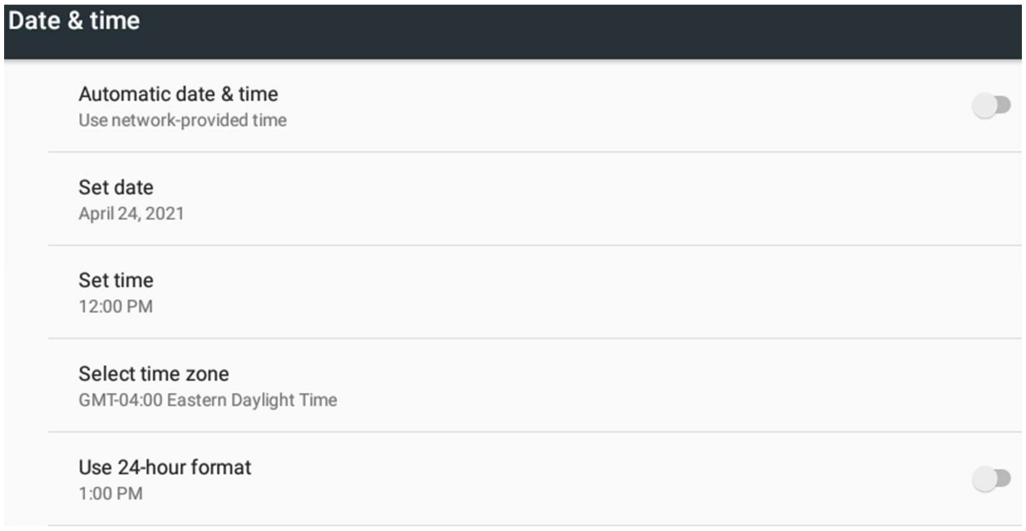
Notifications

Open by default
No defaults set

Figure 21

Step 5: Triggering the attacking code

We run our mock location application and select the location New York and then we change the time on the android device (to 12 PM) to trigger our malicious code.



Date & time

Automatic date & time
Use network-provided time

Set date
April 24, 2021

Set time
12:00 PM

Select time zone
GMT-04:00 Eastern Daylight Time

Use 24-hour format
1:00 PM

Figure 22

Step 6: Track Victim's location

We then load repackagingattacklab.com in our Ubuntu VM and check that the location of the victim is shown as New York. We then change the location on the mock location application to Agra and we see the location is reflected successfully.

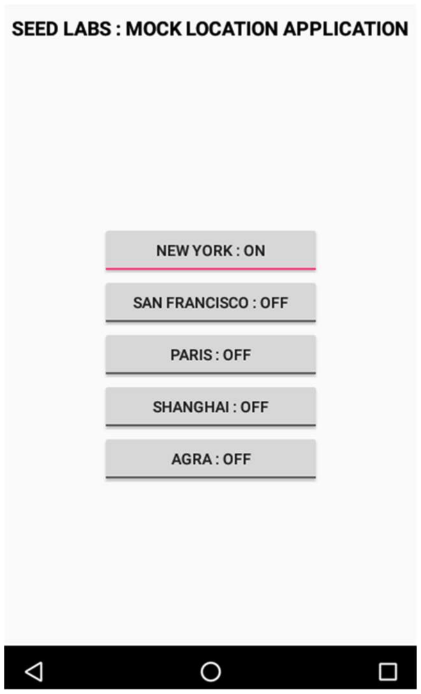


Figure 23

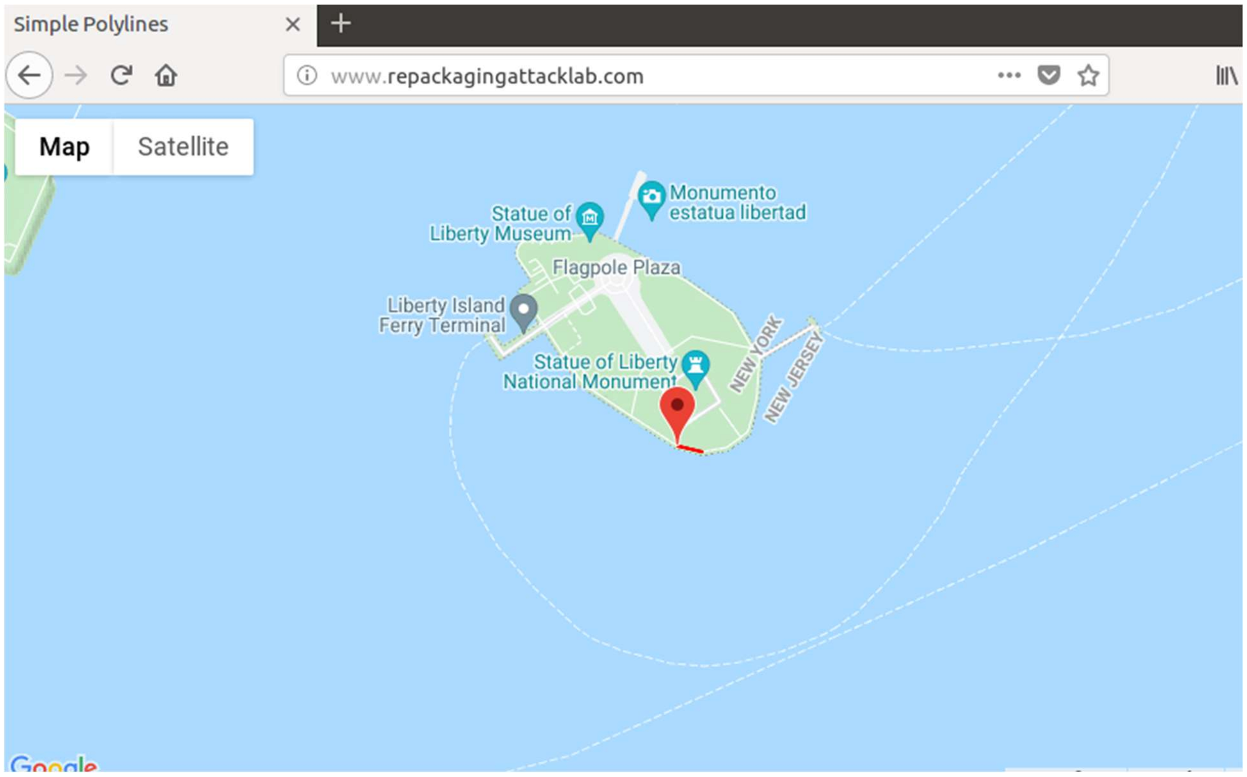

Simple Polylines
www.repackagingattacklab.com
Map
Satellite
Monumento estatua libertad
Statue of Liberty Museum
Flagpole Plaza
Liberty Island Ferry Terminal
Statue of Liberty National Monument
NEW YORK
NEW JERSEY
Google

Figure 24

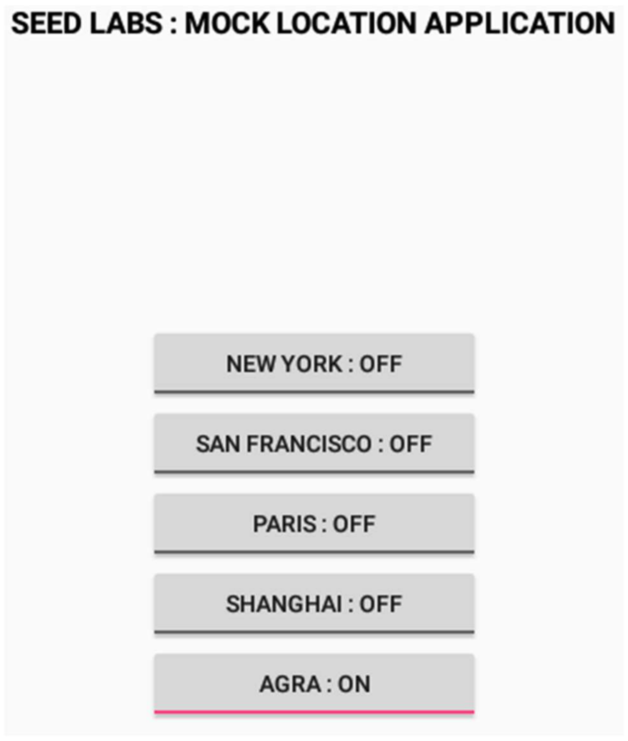


Figure 25

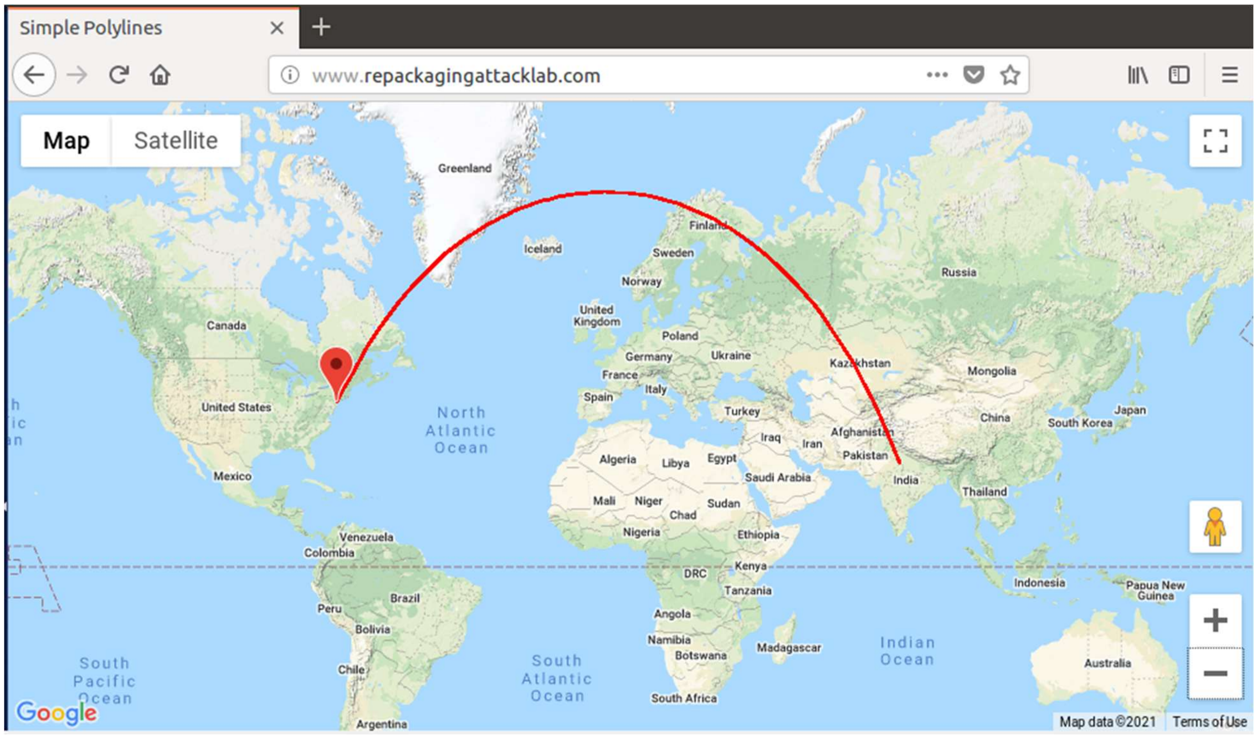


Figure 26