

Owen Bartels, Sebastian Medina

Dr. Ram Basnet

CSCI 465

December 4<sup>th</sup>, 2023

## Android Repackaging Attack Lab Report

### **Introduction**

The Android Repackaging Attack Lab provides a comprehensive hands-on experience in understanding one of the most prevalent security threats in the mobile application ecosystem: repackaging attacks. This lab is designed to impart practical knowledge about how attackers can maliciously modify a legitimate Android application, repackage it, and potentially redistribute the tampered application to unsuspecting users. The primary objective of this lab is not only to understand the technicalities of such attacks but also to appreciate the importance of mobile security measures and ethical considerations in software development and distribution.

Repackaging attacks represent a substantial security risk, primarily because they frequently remain undetected by end-users. Through the process of reverse-engineering a legitimate application, embedding harmful code, and subsequently repackaging it, attackers are capable of illicitly accessing user data and device resources, or potentially compromising the device's integrity. The primary goal of this lab is to enlighten aspiring software developers, security aficionados, and IT experts about these hazards. It aims to empower them to create more secure software solutions and implement robust defenses to thwart such attacks.

## Lab Environment Setup

The lab was conducted in a controlled virtual environment comprising two primary virtual machines (VMs):

1. **SEEDAndroid VM:** This VM is an emulation of an Android device, providing a realistic platform for testing the repackaged applications. It simulates the end-user environment where the modified app would be installed and run.
2. **SEEDUbuntu VM:** Running Ubuntu 16.04, this VM serves as the development and attack simulation environment. It is equipped with all the necessary tools and software required for the lab exercises, including the disassembling and reassembling of APK files, code injection, and app signing.

## Tools and Software Used:

Several tools were utilized throughout the lab to perform various tasks associated with Android repackaging attacks:

- **Android Debug Bridge (ADB):** A versatile command-line tool that allows communication with the Android VM for installing, debugging, and managing applications.
- **APKTool:** A tool for reverse engineering Android apk files. It was used for disassembling APKs to extract Smali code and resources, and reassembling them after modification.
- **Keytool and Jarsigner:** These Java tools were used for generating keystores and signing the repackaged applications, ensuring that they could be installed on the Android VM.

## Task 1: Obtain and Install an Android App (APK)

### Objective:

The primary objective of Task 1 was to familiarize ourselves with the process of obtaining and installing an Android application package (APK). This task is fundamental in understanding the initial steps an attacker might take in a repackaging attack scenario, where a legitimate app is first acquired before being modified.

### Procedure:

1. **APK Acquisition:** We chose a simple Android application, RepackagingLab.apk, which served as the base app for the repackaging attack. This app was either downloaded from a legitimate source or could be a self-developed app in a real-world scenario.
2. **Installation Using ADB:** Utilizing the Android Debug Bridge (ADB) tool, we connected to the SEEDAndroid VM and installed the APK. The process involved the following commands:

- Establishing a connection to the Android VM: adb connect <android\_vm\_ip>

```
[12/03/23]seed@VM:~$ adb connect 10.0.2.4
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.0.2.4:5555
```

- Installing the APK on the VM: adb install RepackagingLab.apk

```
[12/03/23]seed@VM:~$ adb install Downloads/RepackagingLab.apk
1223 KB/s (1421095 bytes in 1.134s)
Failure [INSTALL_FAILED_ALREADY_EXISTS: Attempt to re-install com.mobiseed.repackaging without first uninstalling.]
```

### Learning Outcomes:

- Gained practical experience in handling APK files and understanding their role as the distributable unit in the Android ecosystem.
- Learned the usage of ADB for installing applications on an Android device, a crucial skill for any Android developer or security professional.

### **Challenges Encountered:**

- Initially faced connectivity issues with ADB, which were resolved by ensuring both the host (SEEDUbuntu VM) and the target (SEEDAndroid VM) were on the same network and properly configured.

### **Task 2: Disassemble Android App**

#### **Objective:**

**Task 2 aimed at dissecting the internal structure of an Android app by disassembling the APK file. This step is critical in a repackaging attack to understand how the app is built and to identify areas where malicious code can be injected.**

#### **Procedure:**

1. **Disassembling the APK:** Using APKTool, We disassembled RepackagingLab.apk with the command `apktool d RepackagingLab.apk`. This process unpacked the APK and converted its contents into a human-readable format, including the Smali code and XML resource files.

```
[12/03/23]seed@VM:~$ apktool d Downloads/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...
```

2. **Exploring the App Structure:** We navigated through the disassembled app's directory, examining key components like the AndroidManifest.xml, resource files in the res directory, and the Smali code in the smali folder.

#### **Learning Outcomes:**

- Developed an understanding of the APK structure, including the role of AndroidManifest.xml and how Android apps are compiled into Dalvik Executable (DEX) files.
- Learned about Smali, the assembly language used by the Dalvik VM, which is crucial for modifying app behavior in a repackaging attack.

#### **Challenges Encountered:**

- The complexity of Smali code was initially challenging, requiring careful study to understand its syntax and structure. However, this was an invaluable learning experience in Android app reverse engineering.

### **Task 3: Inject Malicious Code**

#### **Objective:**

The objective of Task 3 was to gain hands-on experience in the core aspect of a repackaging attack: injecting malicious code into an existing Android application. This task is crucial for understanding how attackers can embed harmful functionalities into benign apps.

### Procedure:

- 1. Code Preparation:** We prepared a piece of malicious code intended to demonstrate the potential harm of a repackaging attack. In this case, the code was designed to perform a specific unauthorized action (e.g., accessing and altering user data).
- 2. Injecting the Code:** We located the appropriate place within the disassembled app's Smali code to inject the malicious code. This required careful consideration to ensure the code would execute as intended without disrupting the app's original functionality.
- 3. Modifying AndroidManifest.xml:** We updated the AndroidManifest.xml file to declare any necessary permissions required by the malicious code and to register the new components introduced by it.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="c$
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <application android:allowBackup="true" android:debuggable="true" android:is$
        <activity android:label="@string/app_name" android:name="com.mobiseed.r$
            <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>
```

### Learning Outcomes:

- Understood the process and implications of altering an app's behavior through code injection.
- Learned how Android apps manage permissions and components through the AndroidManifest.xml file.

#### **Challenges Encountered:**

- Ensuring that the injected code was correctly formatted and placed in the Smali code was challenging, requiring a detailed understanding of the app's structure and the Smali syntax.


#### **Task 4: Repackage Android App with Malicious Code**

##### **Objective:**

- Task 4 focused on repackaging the modified app into a new APK file, a critical step in the repackaging attack process. This task demonstrates how an attacker can reassemble an app after injecting malicious code, making it ready for distribution.

##### **Procedure:**

- **Reassembling the APK:** Using APKTool, we reassembled the modified app into a new APK file with the command `apktool b RepackagingLab -o RepackagingLab2.apk`.



**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 repackaging 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.**

- **Signing the APK:** We signed the newly created APK using keytool to generate a keystore and jarsigner to sign the APK. This step is essential as Android requires all apps to be signed before they can be installed.

```
[12/03/23]seed@VM:~$ jarsigner -keystore mykey.keystore RepackagingLab/dist/RepackagingLab.apk owen
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 (2024-03-02) or after any future revocation date.
```



- **Learning Outcomes:**

- Gained practical skills in reassembling and signing APK files, which are essential steps in app development and distribution.
- Understood the importance of app signing in Android's security model.

- **Challenges Encountered:**

- Initially faced issues with the APK reassembly process due to XML syntax errors, which underscored the importance of meticulous attention to detail in modifying app files.

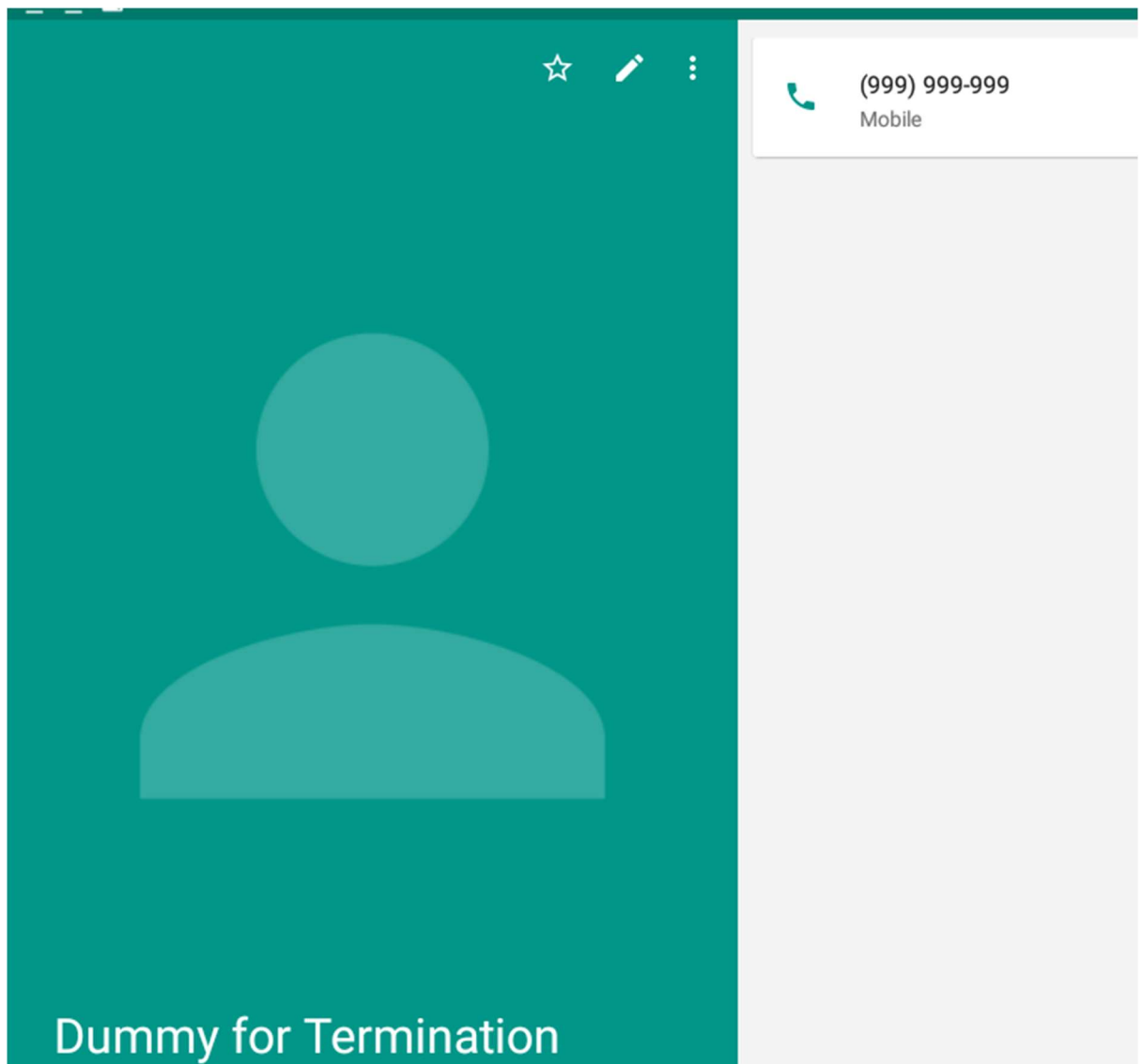
### **Task 5: Install the Repackaged App and Trigger the Malicious Code**

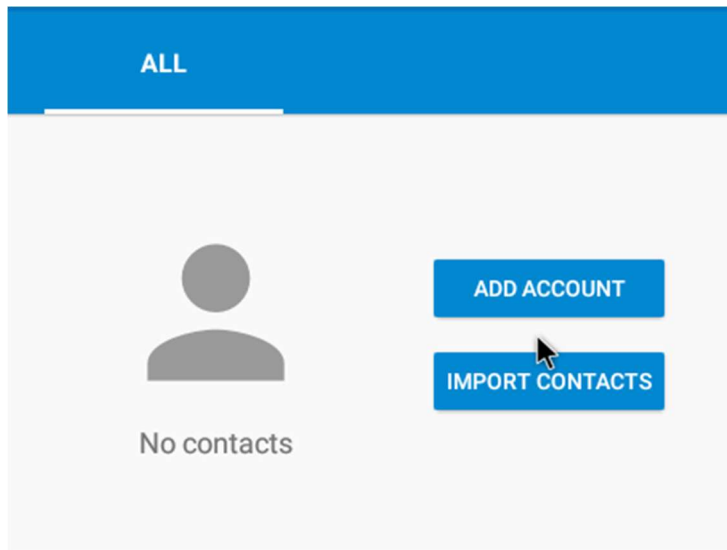
#### **Objective:**

The objective of Task 5 was to install the repackaged app onto the Android VM and validate the functionality of the injected malicious code. This step is crucial in understanding the final stage of a repackaging attack, where the modified app interacts with an actual device environment.

#### **Procedure:**

1. **Installing the Modified App:** We used ADB to install the newly repackaged and signed APK on the SEEDAndroid VM. This involved ensuring that any previous version of the app was uninstalled to avoid signature conflicts.
2. **Triggering the Malicious Code:** After installation, we executed the app to observe the behavior of the injected malicious code. Depending on the nature of the code (e.g., a broadcast receiver listening for a specific system event), we performed actions to trigger it.





### **Learning Outcomes:**

- Experienced the process of deploying an app in a test environment, mimicking the way users would download and install apps from app stores.
- Understood the real-world implications of repackaging attacks, particularly how seemingly benign apps can behave maliciously post-modification.

### **Challenges Encountered:**

- Faced challenges in ensuring the malicious code was triggered as intended, which required a thorough understanding of Android's event handling and app lifecycle.

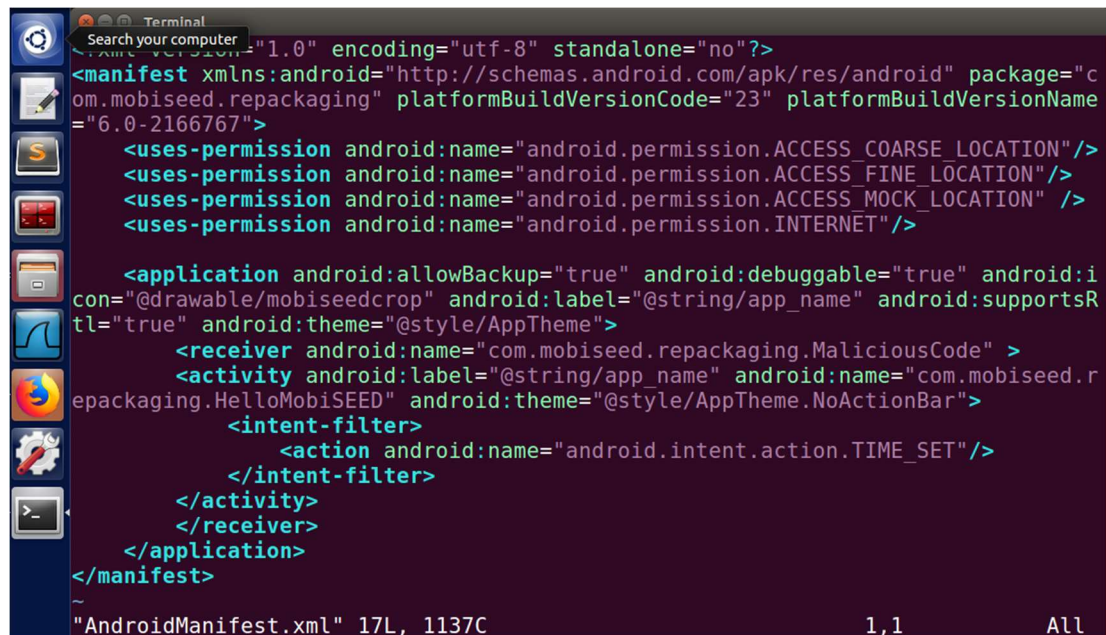
### **Task 6: Using Repackaging Attack to Track Victim's Location**

#### **Objective:**

Task 6 aimed to explore a more advanced repackaging attack scenario where the malicious code was designed to track the victim's location. This task highlights the potential for severe privacy breaches in repackaging attacks.

## Procedure:

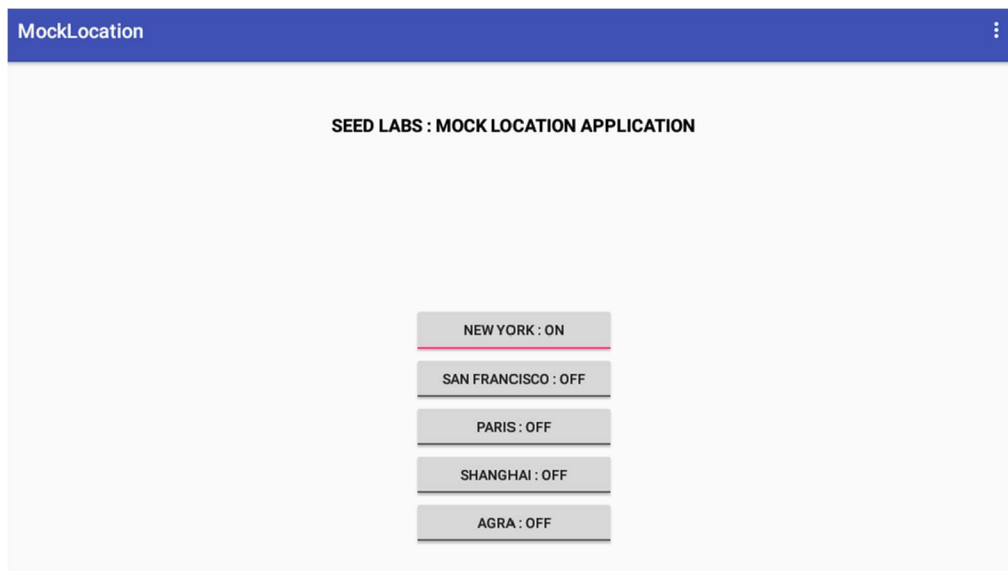
1. **Setting Up Mock Locations:** To simulate location tracking on the Android VM, we set up mock locations. This was crucial since the VM did not have GPS hardware.
2. **Repackaging with Location Tracking Code:** We followed the same steps as in previous tasks to inject, reassemble, and sign an app, this time with code capable of tracking and transmitting the device's location.

A screenshot of a code editor showing the content of an AndroidManifest.xml file. The editor has a dark theme with a sidebar on the left containing icons for various development tools. The main area displays the XML code for the manifest, which includes permissions for location access and internet, and an activity with an intent filter for a TIME\_SET action. The status bar at the bottom shows the file name, line numbers, and a search icon.

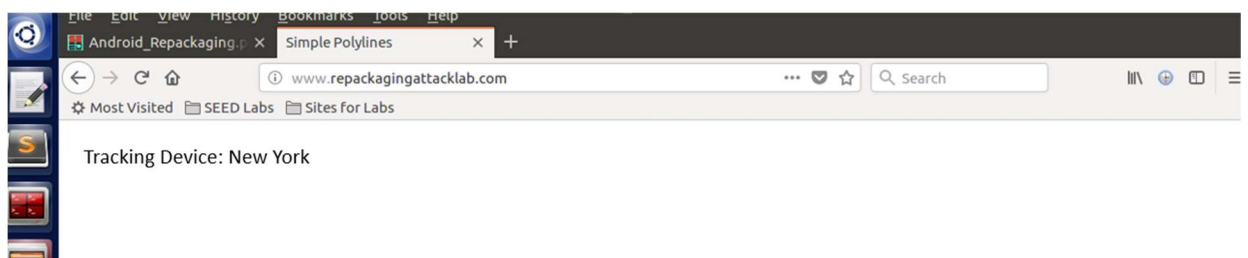
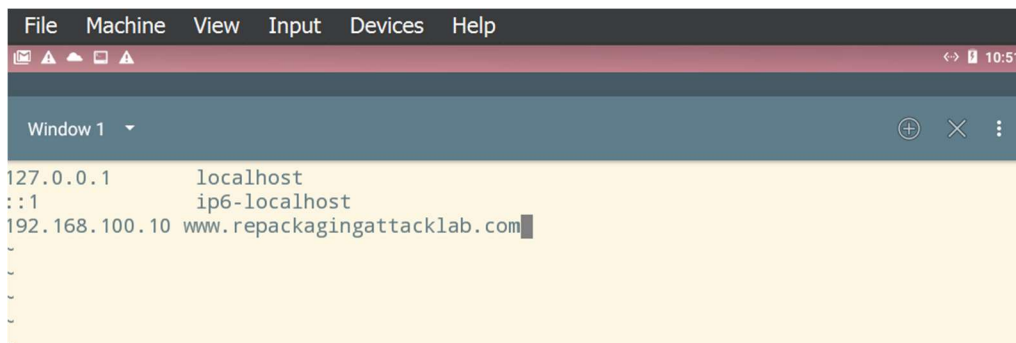
```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.mobiseed.repackaging" platformBuildVersionCode="23" platformBuildVersionName="6.0-2166767">
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>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

3. **DNS Configuration for Data Exfiltration:** We modified the DNS settings to redirect network traffic to a controlled server on the SEEDUbuntu VM, simulating an attacker's server receiving the tracked location data.



4. **Testing and Observing the Attack:** After installing the modified app, we simulated location changes and observed how the app transmitted this data to the designated server.



**Learning Outcomes:**

- Gained insights into how location data can be stealthily extracted from a device, emphasizing the importance of safeguarding sensitive permissions in app development.
- Learned about DNS manipulation and its role in data exfiltration in cyber-attacks.

#### Challenges Encountered:

- Encountered initial difficulties in correctly setting up the mock location and DNS redirection, which required careful network configuration.

### Conclusion

The Android Repackaging Attack Lab has been an enlightening journey into the world of mobile application security, specifically focusing on the vulnerabilities and risks associated with repackaging attacks on Android devices. Through a series of structured tasks, this lab provided a hands-on experience in understanding the entire lifecycle of a repackaging attack, from obtaining and disassembling an APK to injecting malicious code, repackaging, and observing the behavior of the modified app.

Key takeaways from this lab include:

1. **Technical Proficiency:** The lab enhanced our technical skills in using tools like APKTool, ADB, and keytool/jarsigner. We gained practical experience in reverse engineering Android apps, understanding their structure, and manipulating their behavior.
2. **Security Awareness:** The tasks underscored the ease with which legitimate apps can be tampered with and redistributed. This has heightened our awareness of the potential security threats in the apps we use daily and the importance of downloading apps from trusted sources.

3. **Ethical Implications:** Engaging in activities like code injection and repackaging highlighted the ethical responsibilities of software developers and security professionals. It emphasized the need for ethical conduct and the implementation of robust security measures in app development.
4. **Challenges and Problem-Solving:** The lab presented various challenges, from technical issues like XML syntax errors and ADB connectivity problems to conceptual challenges in understanding Smali code. Overcoming these obstacles was a valuable exercise in problem-solving and critical thinking.
5. **Future Applications:** The knowledge and skills acquired in this lab are not only crucial for our careers in cybersecurity but also serve as a foundation for further exploration into mobile security, ethical hacking, and secure app development.