Can ÖZKAN

Introduction to Android Application Reverse Engineering

1) Introduction

Android is an open source, Linux based operating system developed by Google. Due to the fact that it is open source and user friendly, it has become one of the most popular OS in the mobile world. Therefore, android applications become the target of malicious people. Software crackers generally try to figure out software flaws in order to find vulnerabilities and other valuable information such as license key so that they take advantage of the application. It can be achieved by reversing the mobile application, revieweing its source code or just a classic penetration test.

In this writing, I will try to find out the valuable information on the android application named Insecure Bank v2 by performing mobile reverse engineering. Currently, reverse engineering of Android applications is much easier than on other architectures on account of the high level but simple byte code language used. [1]

Insecure Bank v2 is made for security enthusiasts and developers to learn the Android insecurities by testing this deliberately vulnerable application. [2]

Santoku OS [3] is prepared for mobile security testing and it contains tools for Android pentesting / reversing / forensics. Tools we will deal with include but not limited to APKTool, dex2jar, JD-GUI and smali/baksmali. APKTool [4] is a tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications; it makes possible to debug smali code step by step. Dex2Jar [5] is a freely available tool to work with Android ".dex" and Java ".class" files.

2) Introduction to Santoku OS

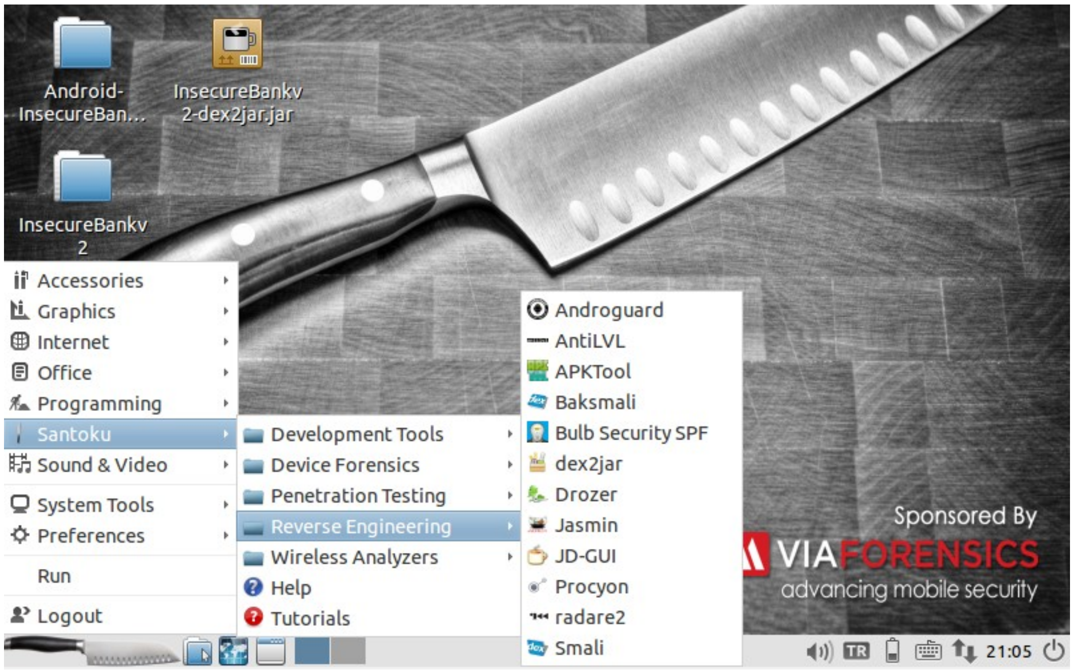
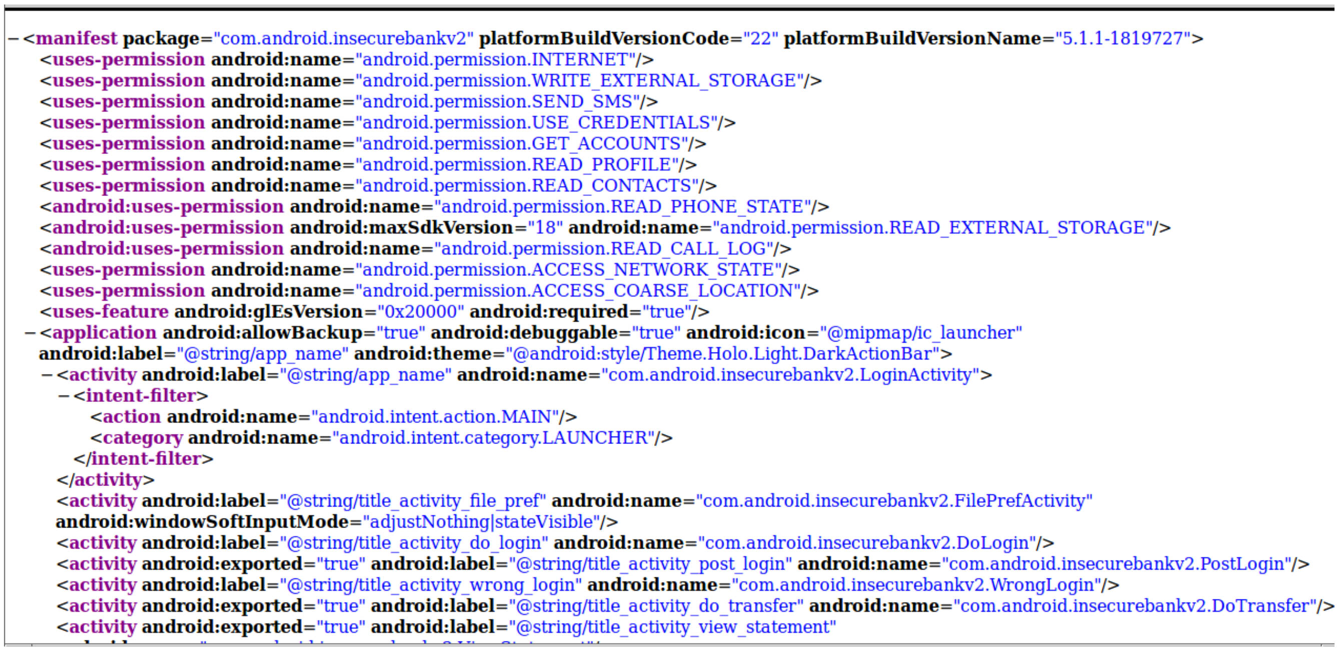


Figure 1

As seen from Figure 1, Santoku OS contains necessary tools in order to perform mobile development, forensics, penetration testing, reverse engineering etc.

3) Reversing InsecureBank v2 Application

First, I use apktool with d option so that I obtain application's resource such as images etc and AndroidManifest.xml. With the help of AndroidManifest.xml, I can learn which permissions the application wants. For privacy concerns, we can delete some of them such as ads, etc, and then re-compile the application. If the application has proper error-handling, you as a user will use the application in an increased privacy level. Otherwise, the application will crush.

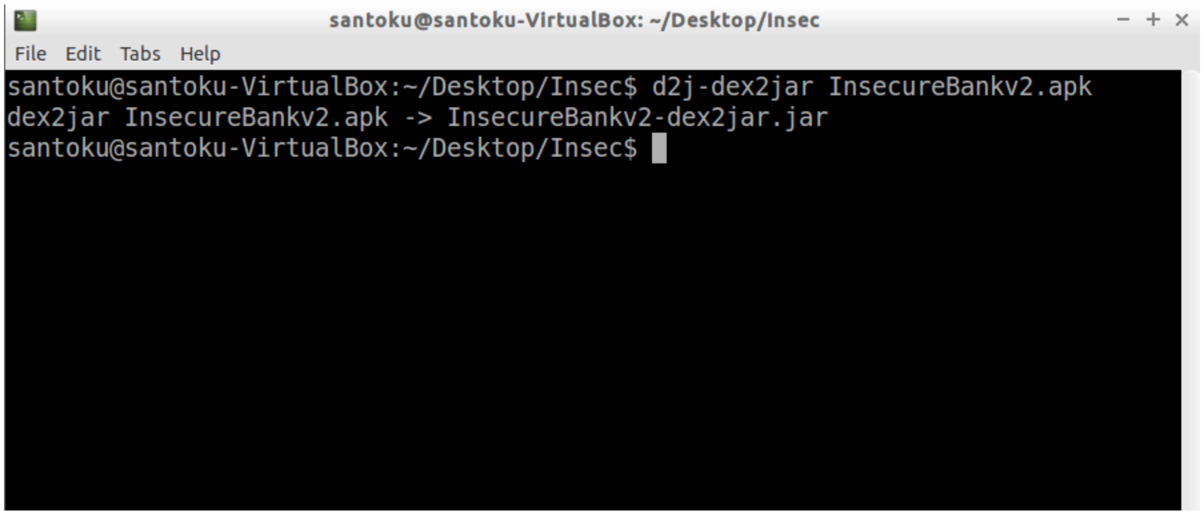


```
– <manifest package="com.android.insecurebankv2" platformBuildVersionCode="22" platformBuildVersionName="5.1.1-1819727">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.USE_CREDENTIALS"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.READ_PROFILE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <android:uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <android:uses-permission android:maxSdkVersion="18" android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <android:uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-feature android:glEsVersion="0x20000" android:required="true"/>
  – <application android:allowBackup="true" android:debuggable="true" android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name" android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
    – <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
      – <intent-filter>
          <action android:name="android.intent.action.MAIN"/>
          <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
      </activity>
      <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity"
      android:windowSoftInputMode="adjustNothing|stateVisible"/>
      <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
      <activity android:exported="true" android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin"/>
      <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
      <activity android:exported="true" android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer"/>
      <activity android:exported="true" android:label="@string/title_activity_view_statement"
```

Figure 2

As seen from Figure 2, you can learn which permissions the application uses and what activities the application has. Activities and intents are of great importance regarding android development process and figuring out inner workings of Android application.

Secondly, on account of the fact that Android APK contains META-INF directory, lib, assets, AndroidManifest.xml, classes.dex and resources.arsc, I will use dex2jar so that I will be able to read classes.dex. The core feature of Dex2Jar is to convert the classes.dex file of an APK to classes.jar or vice versa. So, it is possible to view the source code of an Android application using any Java decompiler, and it is completely readable.



```
santoku@santoku-VirtualBox:~/Desktop/Insec$ d2j-dex2jar InsecureBankv2.apk
dex2jar InsecureBankv2.apk -> InsecureBankv2-dex2jar.jar
santoku@santoku-VirtualBox:~/Desktop/Insec$
```

Figure 3

Then, .jar file appears on the current directory.

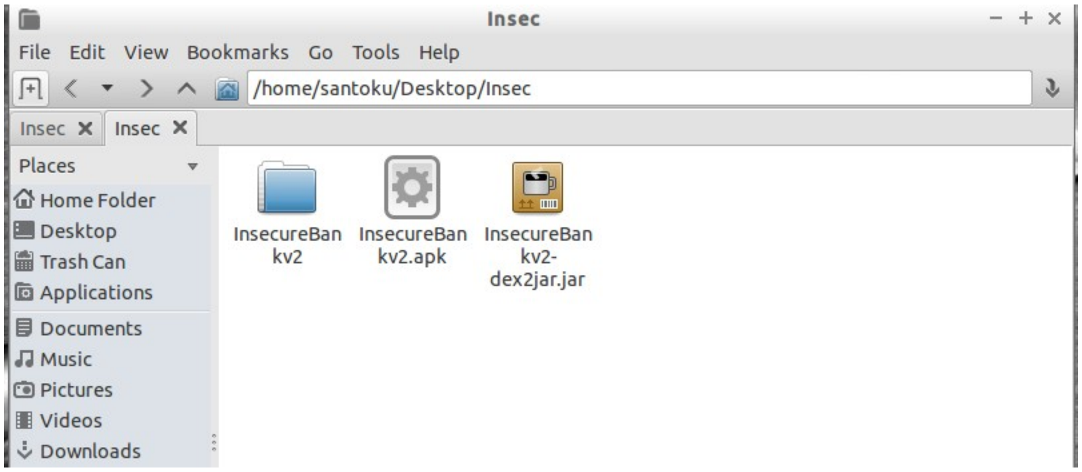
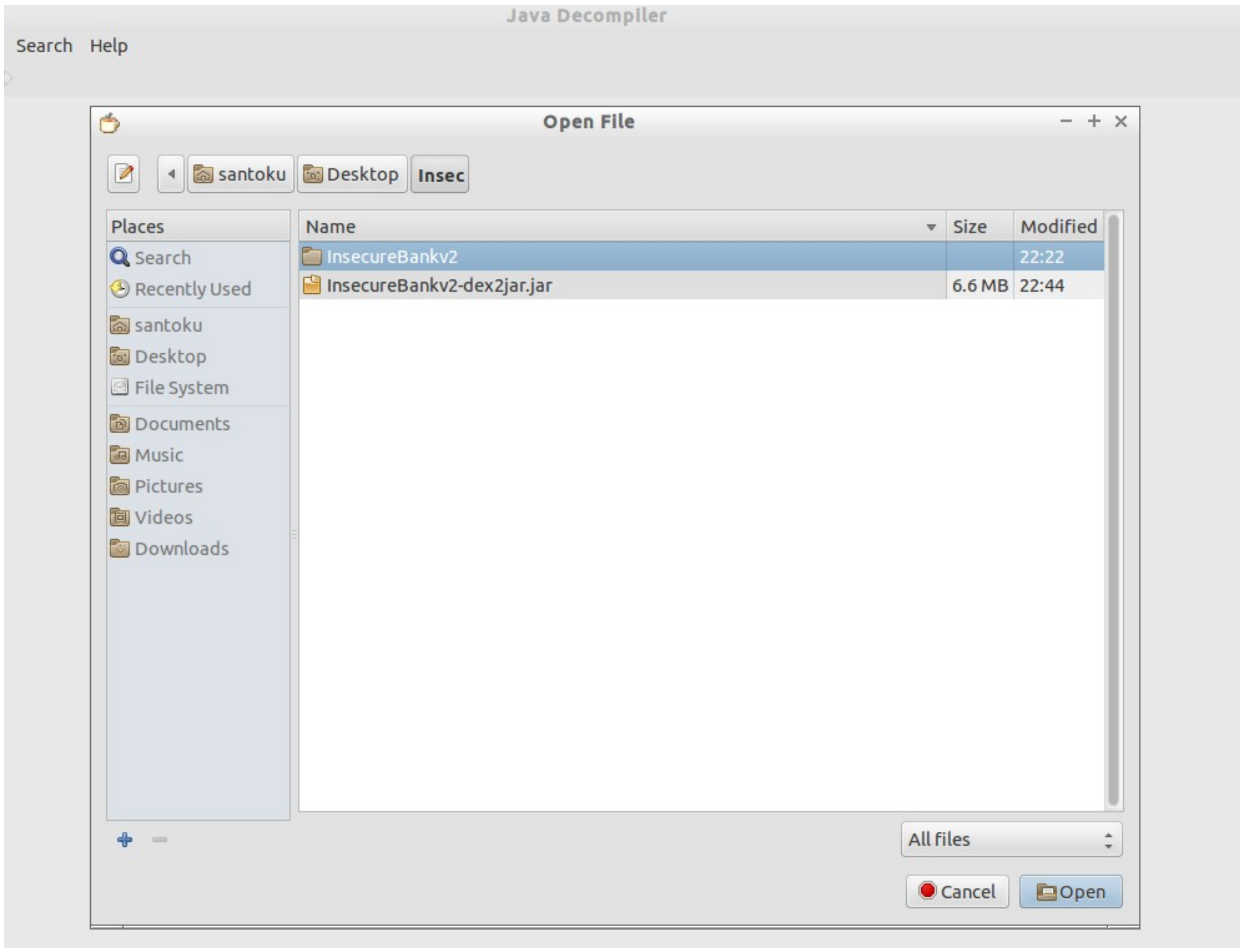


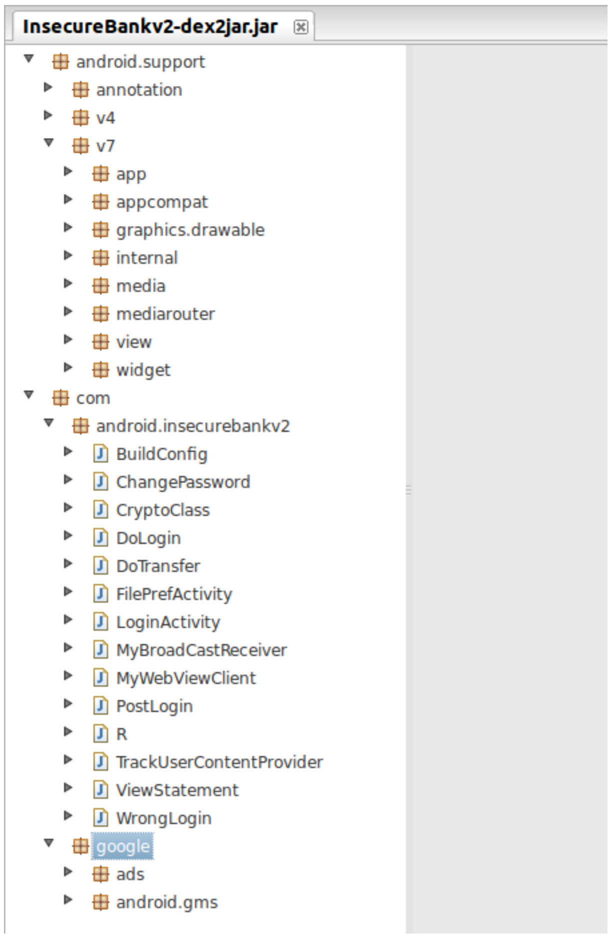
Figure 4

Now, it is time to view the source code of an Android application using any Java decompiler. In this case, I will use JD-GUI. If it is not obfuscated, you will see plain application source code.



Name	Size	Modified
InsecureBankv2		22:22
InsecureBankv2-dex2jar.jar	6.6 MB	22:44

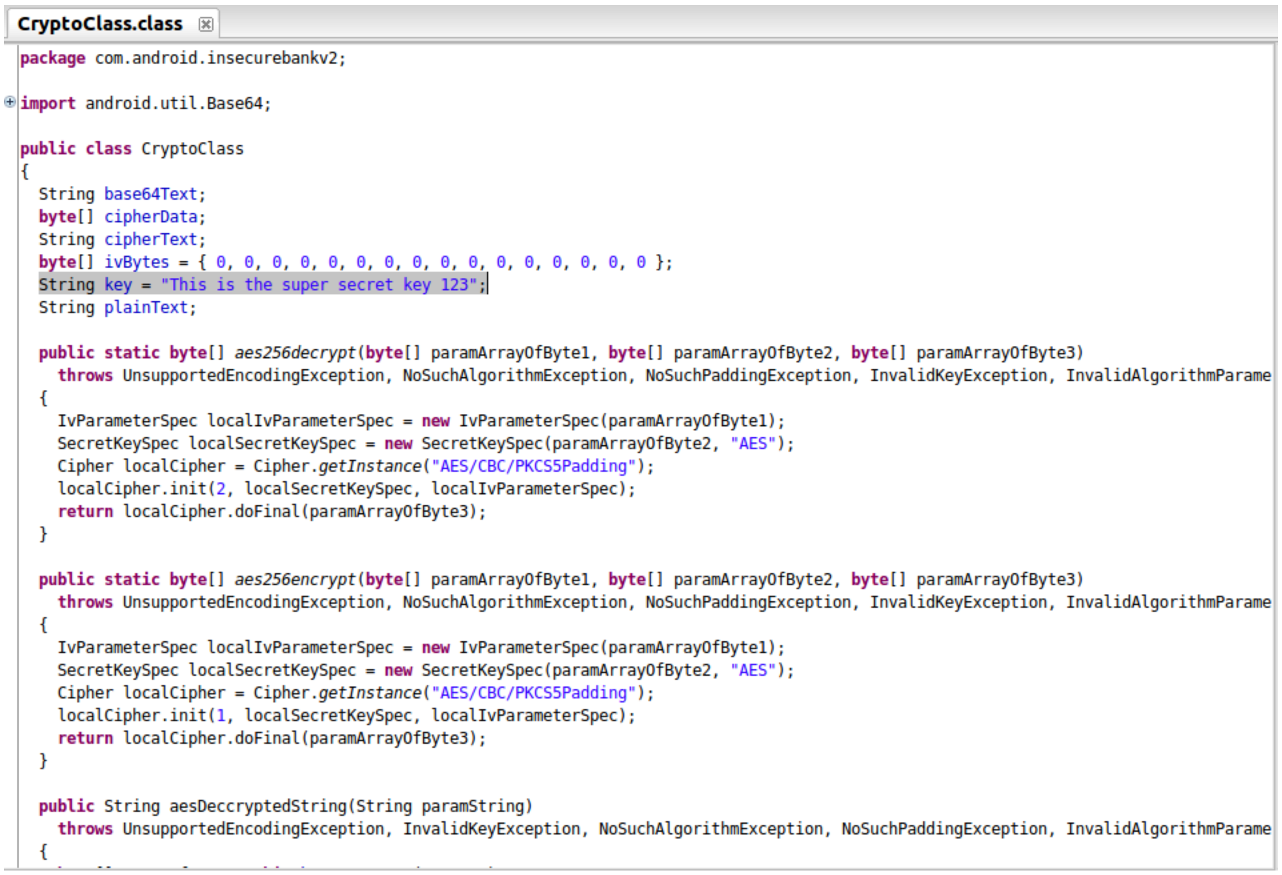
Figure 5



```
InsecureBankv2-dex2jar.jar
▼ android.support
  ▶ annotation
  ▶ v4
  ▼ v7
    ▶ app
    ▶ appcompat
    ▶ graphics.drawable
    ▶ internal
    ▶ media
    ▶ mediarouter
    ▶ view
    ▶ widget
▼ com
  ▼ android.insecurebankv2
    ▶ BuildConfig
    ▶ ChangePassword
    ▶ CryptoClass
    ▶ DoLogin
    ▶ DoTransfer
    ▶ FilePrefActivity
    ▶ LoginActivity
    ▶ MyBroadCastReceiver
    ▶ MyWebViewClient
    ▶ PostLogin
    ▶ R
    ▶ TrackUserContentProvider
    ▶ ViewStatement
    ▶ WrongLogin
  ▼ google
    ▶ ads
    ▶ android.gms
```

Figure 6

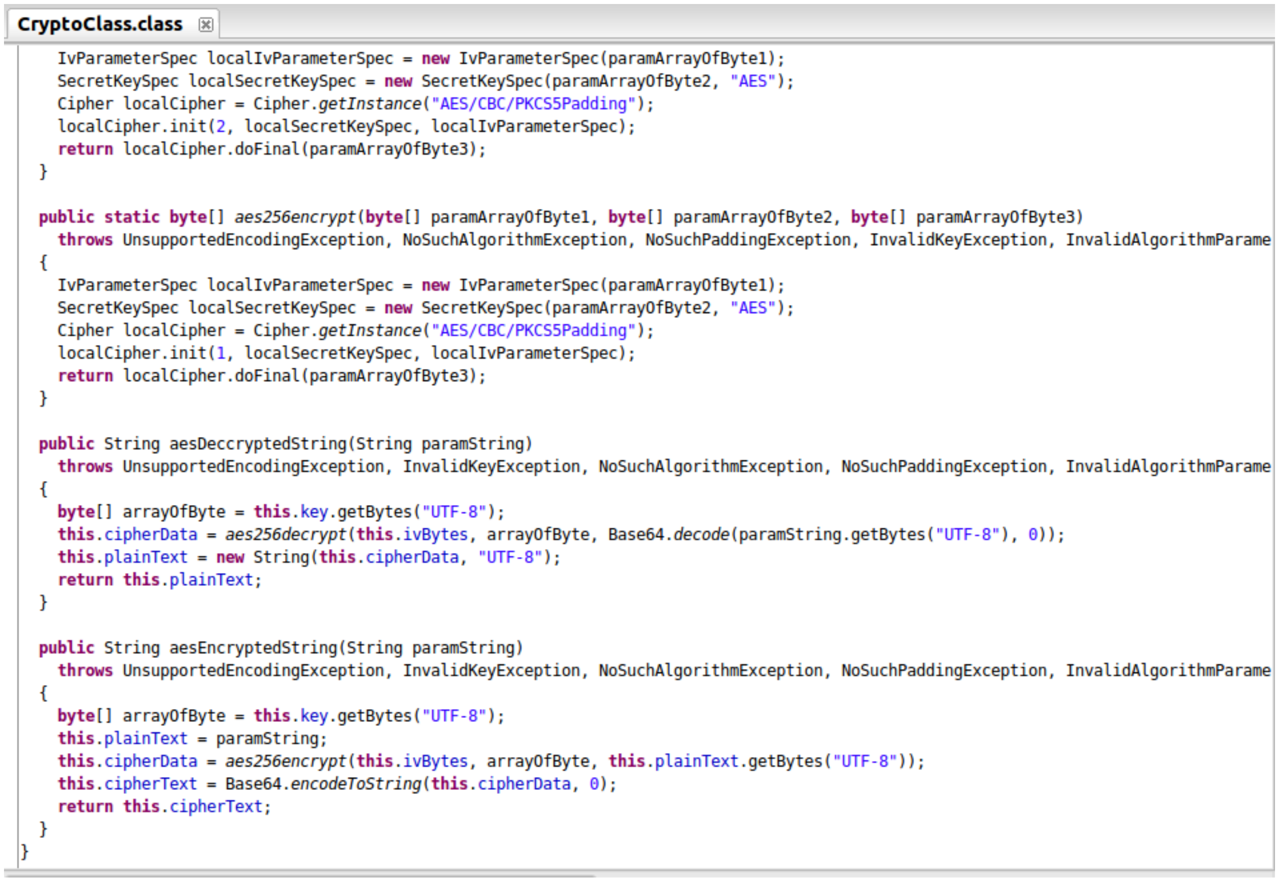
The classes are available for us so as to investigate them. We will take a glance at all the classes in order to understand what the application's features, methods etc.



```
CryptoClass.class
package com.android.insecurebankv2;

import android.util.Base64;

public class CryptoClass
{
  String base64Text;
  byte[] cipherData;
  String cipherText;
  byte[] ivBytes = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
  String key = "This is the super secret key 123";
  String plainText;

  public static byte[] aes256decrypt(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2, byte[] paramArrayOfByte3)
    throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParame
  {
    IvParameterSpec localIvParameterSpec = new IvParameterSpec(paramArrayOfByte1);
    SecretKeySpec localSecretKeySpec = new SecretKeySpec(paramArrayOfByte2, "AES");
    Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    localCipher.init(2, localSecretKeySpec, localIvParameterSpec);
    return localCipher.doFinal(paramArrayOfByte3);
  }

  public static byte[] aes256encrypt(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2, byte[] paramArrayOfByte3)
    throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParame
  {
    IvParameterSpec localIvParameterSpec = new IvParameterSpec(paramArrayOfByte1);
    SecretKeySpec localSecretKeySpec = new SecretKeySpec(paramArrayOfByte2, "AES");
    Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    localCipher.init(1, localSecretKeySpec, localIvParameterSpec);
    return localCipher.doFinal(paramArrayOfByte3);
  }

  public String aesDeccryptedString(String paramString)
    throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParame
  {
```

Figure 7

It is better to look at CryptoClass. There is an interesting field named key. In this application, as you can see, there is no obfuscation, anti-reversing, white-box cryptography techniques at all. For this reason, you can just look at the code and figure out what the application does.



```
CryptoClass.class
    IvParameterSpec localIvParameterSpec = new IvParameterSpec(paramArrayOfByte1);
    SecretKeySpec localSecretKeySpec = new SecretKeySpec(paramArrayOfByte2, "AES");
    Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    localCipher.init(2, localSecretKeySpec, localIvParameterSpec);
    return localCipher.doFinal(paramArrayOfByte3);
  }

  public static byte[] aes256encrypt(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2, byte[] paramArrayOfByte3)
    throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParame
  {
    IvParameterSpec localIvParameterSpec = new IvParameterSpec(paramArrayOfByte1);
    SecretKeySpec localSecretKeySpec = new SecretKeySpec(paramArrayOfByte2, "AES");
    Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    localCipher.init(1, localSecretKeySpec, localIvParameterSpec);
    return localCipher.doFinal(paramArrayOfByte3);
  }

  public String aesDeccryptedString(String paramString)
    throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParame
  {
    byte[] arrayOfByte = this.key.getBytes("UTF-8");
    this.cipherData = aes256decrypt(this.ivBytes, arrayOfByte, Base64.decode(paramString.getBytes("UTF-8"), 0));
    this.plainText = new String(this.cipherData, "UTF-8");
    return this.plainText;
  }

  public String aesEncryptedString(String paramString)
    throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParame
  {
    byte[] arrayOfByte = this.key.getBytes("UTF-8");
    this.plainText = paramString;
    this.cipherData = aes256encrypt(this.ivBytes, arrayOfByte, this.plainText.getBytes("UTF-8"));
    this.cipherText = Base64.encodeToString(this.cipherData, 0);
    return this.cipherText;
  }
}
```

Figure 8

This field is used in aesDecryptedString() and aesEncyptedString() methods. Moreover, those two functions invoke aes256encrpt and aes256decrypt methods. As a result, we can say the fact that we recover the hardcoded AES 256 bit key used in communication.

4) Conclusion

To sum up, bad threat actors can recover and read your application’s corresponding source code unless developed properly. It is recommended that developers use obfuscation, anti-reversing techniques, white-box cryptography and finally follow secure software development best practices in order to strengthen the application’s overall security posture.

5) References

[1] https://www.semanticscholar.org/paper/Securing-Android-Code-Using-White-Box-Cryptography-Anand/798deca80c85217483ed46da34fa3629b2c459a0

[2] https://github.com/dineshshetty/Android-InsecureBankv2

[3] https://santoku-linux.com/about-santoku/

[4] https://github.com/iBotPeaches/Apktool

[5] https://resources.infosecinstitute.com/android-penetration-tools-walkthrough-series-dex2jar-jd-gui-baksmali/#gref