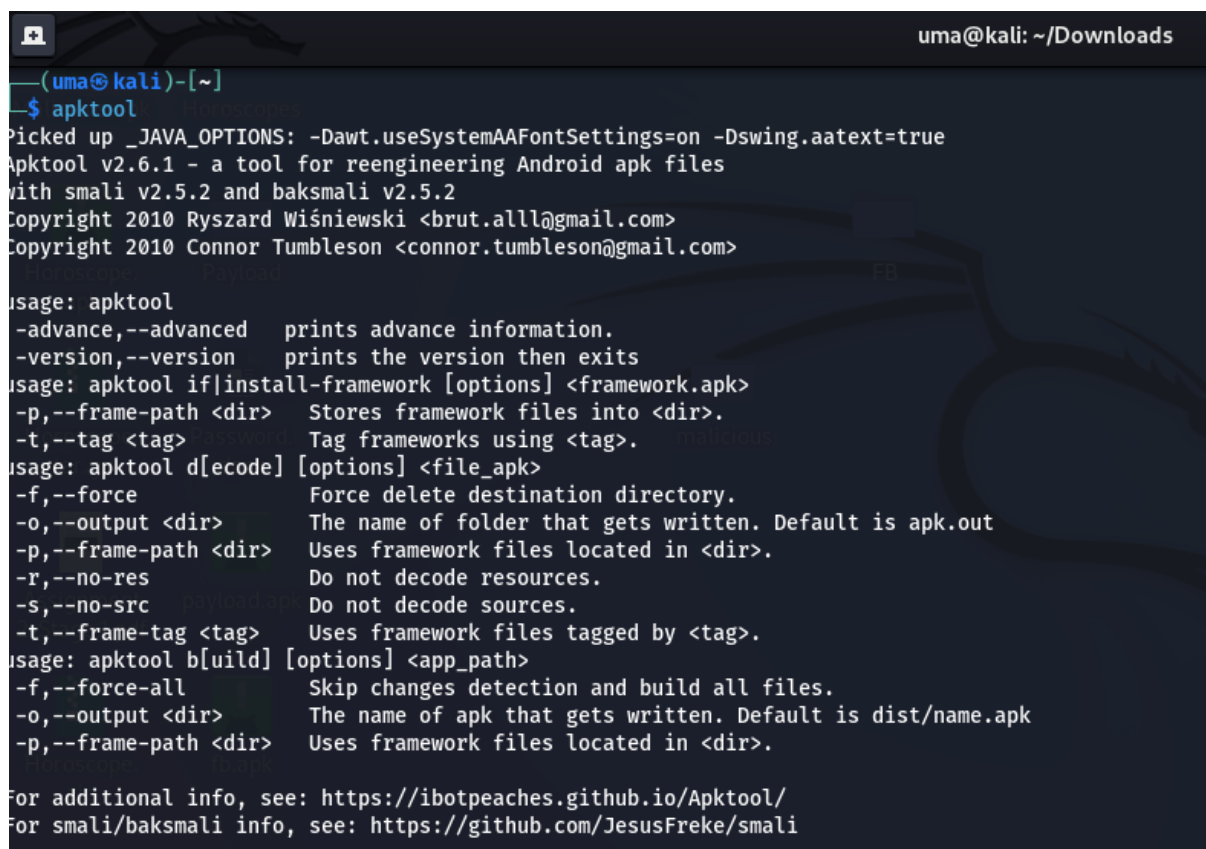


Malicious APK File Analysis

No. 1

1. To open the apktool we use the following command in command prompt- "**apktool**".



```
uma@kali: ~/Downloads
(uma@kali)-[~]
$ apktool
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Apktool v2.6.1 - a tool for reengineering Android apk files
with smali v2.5.2 and baksmali v2.5.2
Copyright 2010 Ryszard Wiśniewski <brut.alll@gmail.com>
Copyright 2010 Connor Tumbleson <connor.tumbleson@gmail.com>

Usage: apktool
  -advance,--advanced      prints advance information.
  -version,--version       prints the version then exits
Usage: apktool if|install-framework [options] <framework.apk>
  -p,--frame-path <dir>   Stores framework files into <dir>.
  -t,--tag <tag>          Tag frameworks using <tag>.
Usage: apktool d[ecode] [options] <file_apk>
  -f,--force              Force delete destination directory.
  -o,--output <dir>       The name of folder that gets written. Default is apk.out
  -p,--frame-path <dir>   Uses framework files located in <dir>.
  -r,--no-res             Do not decode resources.
  -s,--no-src             Do not decode sources.
  -t,--frame-tag <tag>    Uses framework files tagged by <tag>.
Usage: apktool b[uild] [options] <app_path>
  -f,--force-all         Skip changes detection and build all files.
  -o,--output <dir>       The name of apk that gets written. Default is dist/name.apk
  -p,--frame-path <dir>   Uses framework files located in <dir>.

For additional info, see: https://ibotpeaches.github.io/Apktool/
For smali/baksmali info, see: https://github.com/JesusFreke/smali
```

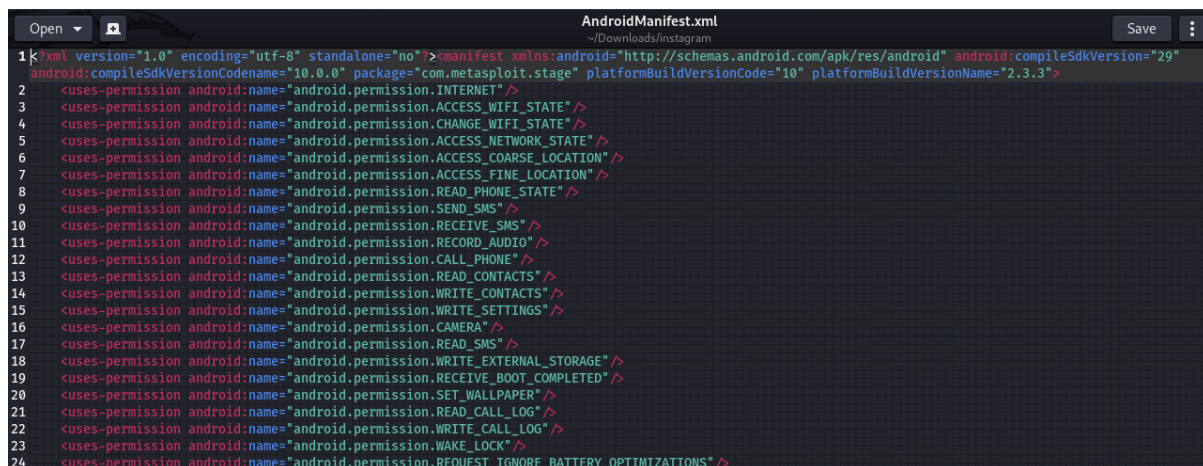
2. After that go to the path where apk file is located. Our apk file is located in the Downloads. Then decompile the apk file using the below command – "**apktool d Instagram.apk**", where d is use to decompile the apk file.

```
(uma@kali)~[~]
$ cd Downloads

(uma@kali)~[~/Downloads]
$ apktool d instagram.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.6.1 on instagram.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/uma/.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...

(uma@kali)~[~/Downloads]
$
```

3. After decompiling the given apk file, the following permissions were granted to the application as observed in “**AndroidManifest.xml**”.



```
AndroidManifest.xml
~/Downloads/instagram
Save

1 <?xml version="1.0" encoding="utf-8" standalone="no"><manifest xmlns:android="http://schemas.android.com/apk/res/android" android:compileSdkVersion="29"
2   android:compileSdkVersionCodename="10.0.0" package="com.metasploit.stage" platformBuildVersionCode="10" platformBuildVersionName="2.3.3">
3   <uses-permission android:name="android.permission.INTERNET" />
4   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
5   <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
6   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
7   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
8   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
9   <uses-permission android:name="android.permission.READ_PHONE_STATE" />
10  <uses-permission android:name="android.permission.SEND_SMS" />
11  <uses-permission android:name="android.permission.RECEIVE_SMS" />
12  <uses-permission android:name="android.permission.RECORD_AUDIO" />
13  <uses-permission android:name="android.permission.CALL_PHONE" />
14  <uses-permission android:name="android.permission.READ_CONTACTS" />
15  <uses-permission android:name="android.permission.WRITE_CONTACTS" />
16  <uses-permission android:name="android.permission.WRITE_SETTINGS" />
17  <uses-permission android:name="android.permission.CAMERA" />
18  <uses-permission android:name="android.permission.READ_SMS" />
19  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
20  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
21  <uses-permission android:name="android.permission.SET_WALLPAPER" />
22  <uses-permission android:name="android.permission.READ_CALL_LOG" />
23  <uses-permission android:name="android.permission.WRITE_CALL_LOG" />
24  <uses-permission android:name="android.permission.WAKE_LOCK" />
25  <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
```

➔ Permissions were granted such as Internet, Location, Phone, Contacts, Settings, Storage and etc.

4. There is only one activity named “**MainActivity**” that is intended by the APK as observed in the strings.xml file. This can be a main triggering activity for the APK that can further invoke other actions.

```
strings.xml
~/Downloads/instagram1/res/values

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string name="app_name">MainActivity</string>
4   <string name="secret">HELLO WORLD!</string>
5 </resources>
```

- When we analyze the AndroidManifest.xml further we observe the startup class for the android APK is **"MainActivity"** as its being bound to the **intent.action.main**.

```
29 <activity android:label="@string/app_name" android:name=".MainActivity" android:theme="@android:style/Theme.NoDisplay">
30   <intent-filter>
31     <action android:name="android.intent.action.MAIN" />
32     <category android:name="android.intent.category.LAUNCHER" />
33   </intent-filter>
34   <intent-filter>
35     <data android:host="my_host" android:scheme="metasploit" />
36     <category android:name="android.intent.category.DEFAULT" />
37     <category android:name="android.intent.category.BROWSABLE" />
38     <action android:name="android.intent.action.VIEW" />
39   </intent-filter>
40 </activity>
```

- ➔ When analyzing the XML, we see the onCreate service implementation would be available in the MainActivity class as mentioned in the **android:name** tag.

- This code might be suspicious and it seems to be obfuscated.

```
b.class - Java Decompiler
File Edit Navigation Search Help

classes-dex2jar.jar x
└─ com.metasploit.stage
   └─ MainActivity.class
   └─ MainBroadcastRece
   └─ MainService.class
   └─ Payload.class
      └─ Payload
         a: byte[]
         b: Context
         c: long
         d: byte[]
         e: String
         f: String
         g: String
         h: Object[]
         Payload()
         a(): void
         a(DataInputStre
         a(DataInputStre
         main(String[]):
         start(Context):
         startContext():
         startInPath(Stri

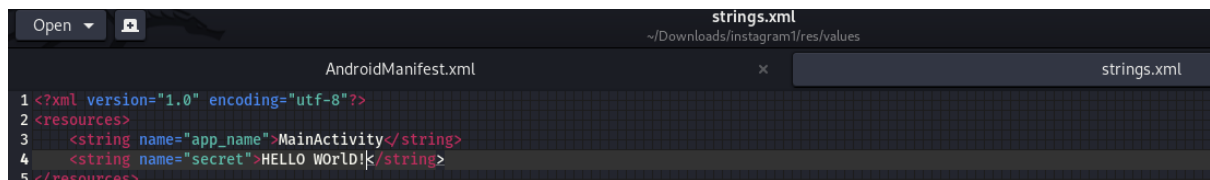
MainService.class x
package com.metasploit.stage;
import java.io.UnsupportedEncodingException;
import java.util.concurrent.TimeUnit;

public final class b {
    private static final long a = TimeUnit.SECONDS.toMillis(1L);

    private static int a(byte[] paramArrayOfbyte, int paramInt) {
        byte b1 = 0;
        int i = 0;
        while (b1 < 4) {
            i |= (paramArrayOfbyte[b1 + paramInt] & 0xFF) << b1 << 3;
            b1++;
        }
        return i;
    }

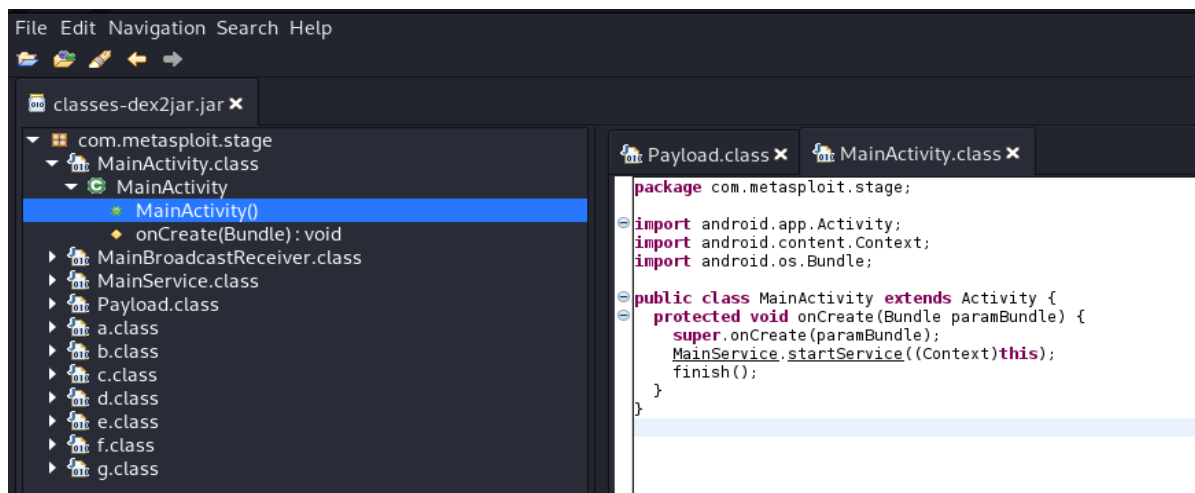
    public static a a(byte[] paramArrayOfbyte) {
        a a = new a();
        a.a = a(paramArrayOfbyte, 0);
        a.b = a * a(paramArrayOfbyte, 12);
        b(paramArrayOfbyte, 16, 16);
        b(paramArrayOfbyte, 32, 16);
        int i = 48;
        int i = i;
```

- The secret code we have found for this application as **"HELLO WORLD!"** as observed in the **"Strings.xml"**.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string name="app_name">MainActivity</string>
4   <string name="secret">HELLO WORLD!</string>
5 </resources>
```

8. The MainActivity class when decompiled with dex2jar on the classes.dex obtained by unzip command on the APK we observe it as follows:



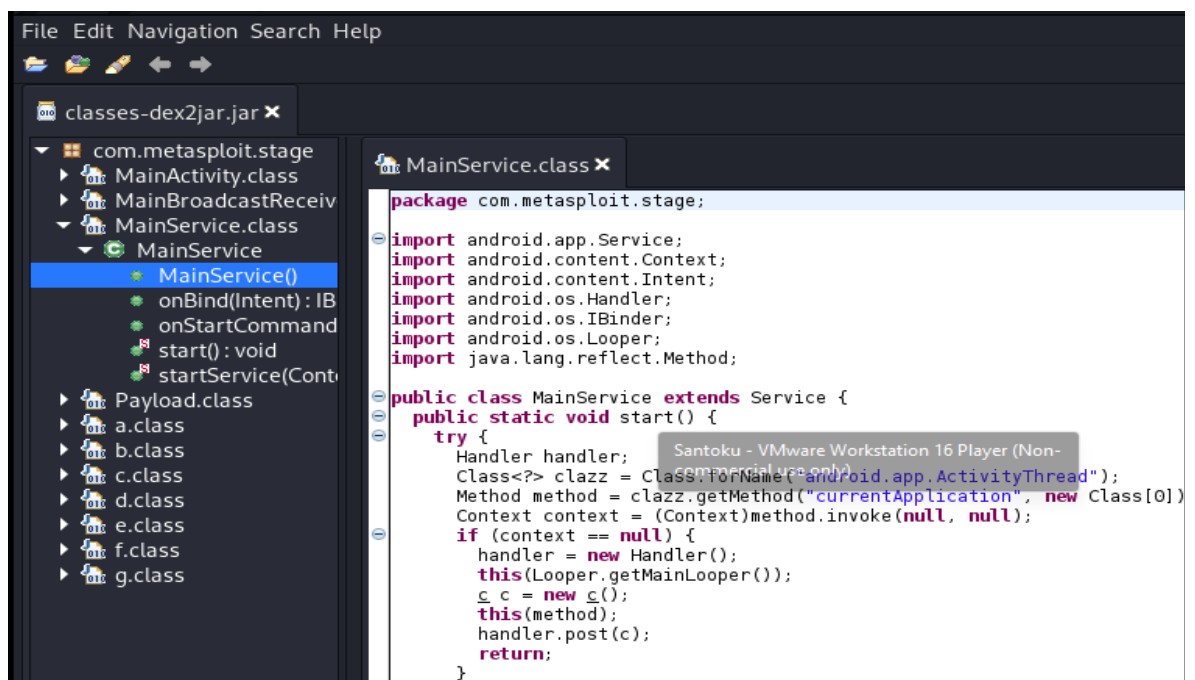
```
package com.metasploit.stage;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;

public class MainActivity extends Activity {
    protected void onCreate(Bundle paramBundle) {
        super.onCreate(paramBundle);
        MainService.startService((Context)this);
        finish();
    }
}
```

➔ Here the **MainActivity** is calling the **MainService** class to start service with the activity context when the Booting of the android is completed within the **onCreate** method.

9. Now let's see the **MainService** class implementation:



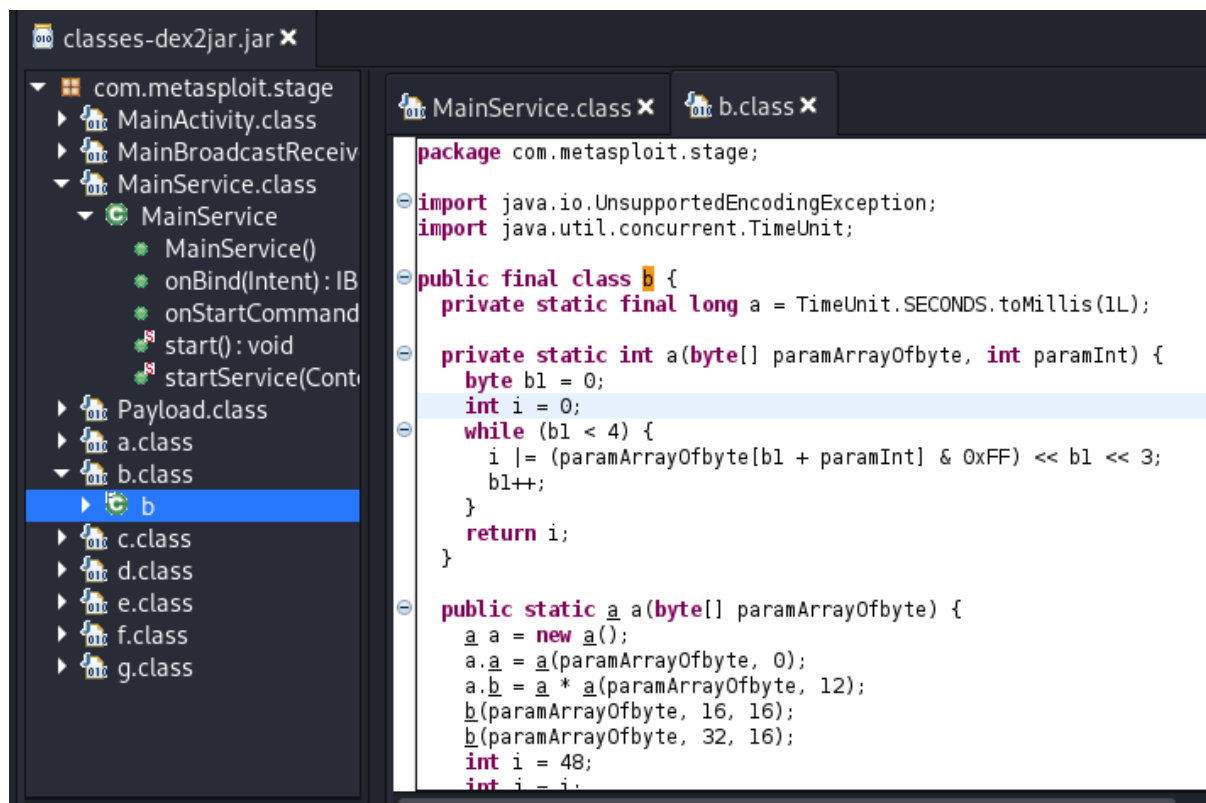
```
package com.metasploit.stage;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Looper;
import java.lang.reflect.Method;

public class MainService extends Service {
    public static void start() {
        try {
            Handler handler;
            Class<?> clazz = Class.forName("android.app.ActivityThread");
            Method method = clazz.getMethod("currentApplication", new Class[0]);
            Context context = (Context)method.invoke(null, null);
            if (context == null) {
                handler = new Handler();
                this(Looper.getMainLooper());
                c c = new c();
                this(method);
                handler.post(c);
                return;
            }
        }
    }
}
```

➔ Here we can see the start method initiates a thread for the current application and as soon as a context is created, we attach the context to the given handler for posting.

10. Inside b.class generated we see as follows:



➔ There are some offsets being used for the http URL under consideration and some operations like **Arrays.copyOfOf** being applied upon these offsets showing a possible scenario of obfuscation.