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**:

   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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MainActivity</string>
</resources>

```

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>
        <intent-filter>
```

When analyzing the XML, we see the onCreate service implementation would be available in the MainActivity class as mentioned in the ***andoid: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 | d.class | e.class | f.class | g.class | MainActivity.class

MainBroadcastReceiver.class | MainService.class

```
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.

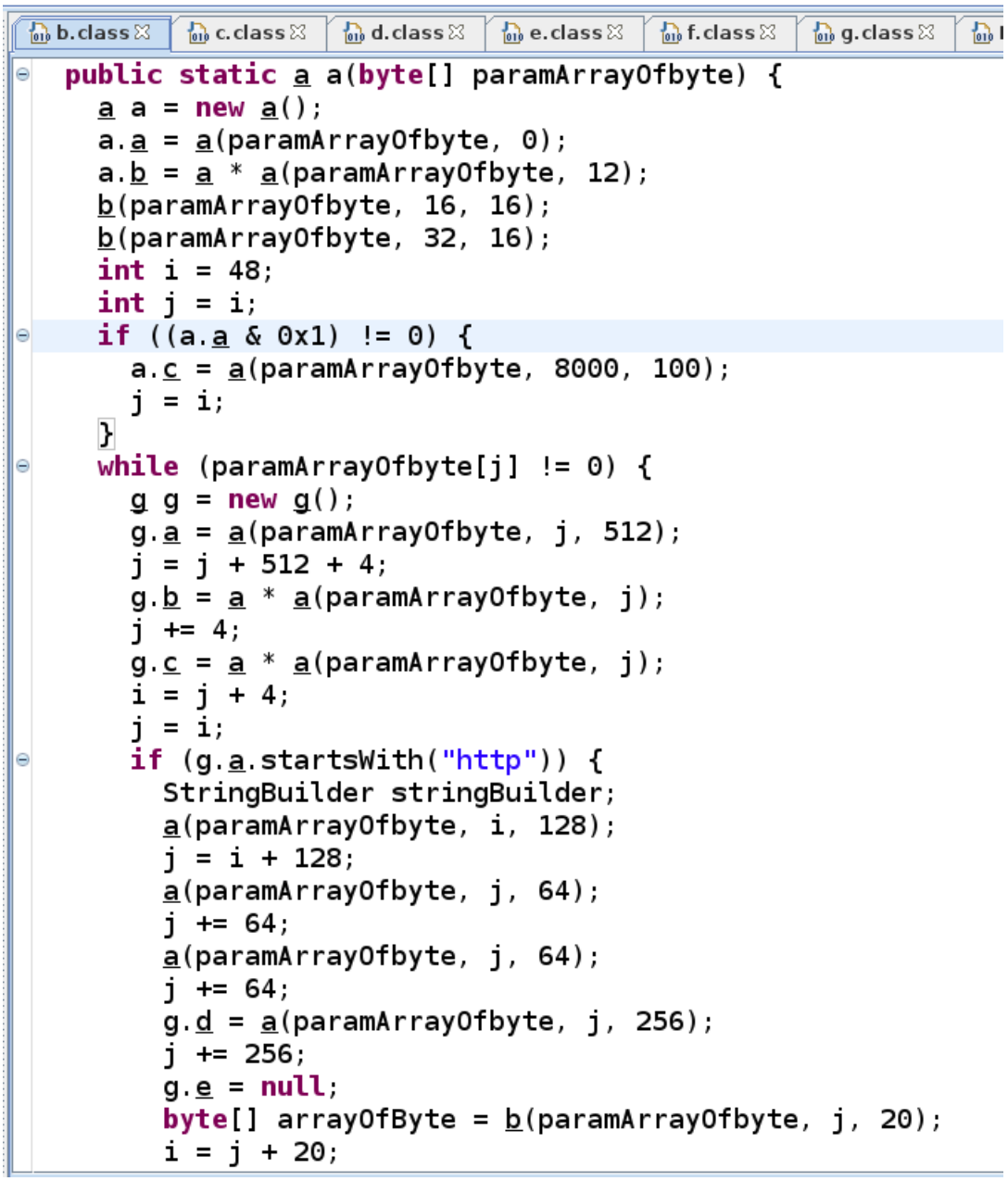
5. Now let's see the ***MainService*** class implementation:

MainBroadcastReceiver.class | MainService.class | Payl

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

public class MainService extends Service {
  public static void start() {
    try {
      c 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:



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.