**Android 插件化开发 之 Hook APK应用并注入代码**

1. 背景和意义

在Android移动端安全领域，我们需要监听我们的手机应用，光写个Android应用是没有用的，最多只能对网络数据的监听。所以必须要在应用里注入自己的代码，并且使用Hook的功能进行监控Android应用。并且在此过程中我们可以衍生出很多项目如：Android插件化开发、热修复等。能做的东西会很多，当然这对Android开发的功底有点大。

2. Java反射相关知识

反射作用:通过类名获取成员变量和方法的返回值以及替换类的成员变量......

部分代码演示:

1)获取 class 对象

Class<?> c = Class.forName("classname");

2)获取函数返回值

//funcname 表示调用方法的名称,c 表示参数的 Class 对象

Method m = c.getMethod("funcname",Class<?>...c);

例如:Method m = c.getMethod("fun",String.class,int.class);

表示调用 fun(String,int);

//如果有返回值,则 invoke 函数返回;

如果是私有函数,还需要加上 m.setAccessible(true);

函数 Object obj = m.invoke(c1.newInstance(),"xiazdong",20);

/ /注:如果是调用静态的方法,则不需要设置对象;

Object obj = m.invoke(null,"xiazdong");

3) 替换成员变量

Field f = c.getDeclaredField("name");

f.setAccessible(true);

//返回 name 属性

//私有属性可见

String name = (String)f.get(Object obj);

f.set(Object obj,String n);

//返回 obj 对象的 name 属性的值

//设置 obj 对象的 name 属性为 n 值;

3. Java 静态代理

意图：

为其他对象提供一种代理以控制这个对象的访问。

适用性：

　　远程代理（ Remote Proxy ）： 为一个对象在不同的地址空间提供局部代表。

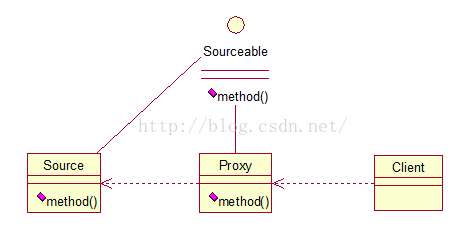
　　虚代理（Virtual Proxy）根据需要创建开销很大的对象。使用一个代理对象作为代表，在真正的需要时进行创建。

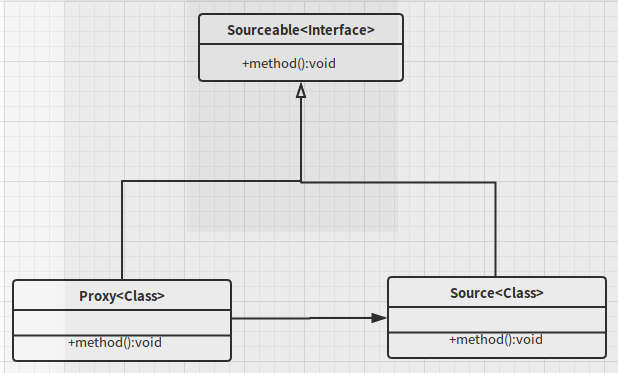
　　保护代理（Protection Proxy）：控制对原始对象的访问。保护代理用于对象应该有不同的访问权限的时候。

　　智能指引（Smart Reference）： 取代了简单的指针，它在访问对象时执行一些附加操作。对指向实际对象的引用计数，这样当该对象没有引用时，可以自动释放它，智能指针当第一次引用一个持久对象时，将它装入内存。

　　 在访问一个实际对象前，检查是否已经锁定了它，以确保其他对象不能改变它

其实每个模式名称就表明了该模式的作用，代理模式就是多一个代理类出来，替原对象进行一些操作，比如我们在租房子的时候回去找中介，为什么呢？因为你对该地区房屋的信息掌握的不够全面，希望找一个更熟悉的人去帮你做，此处的代理就是这个意思。再如我们有的时候打官司，我们需要请律师，因为律师在法律方面有专长，可以替我们进行操作，表达我们的想法。先来看看关系图：





3、代码实现单独文件分析

4、运行结果:

before proxy!

the original method!

after proxy!

3. Java 动态代理

静态代理每个代理类只能为一个接口服务,这样程序开发中必然会产生许多的代理类，所以我们就会想办法可以通过一个代理类完成全部的代理功能,那么我们就需要用动态代理,动态生产代理对象。

在 Java 中 要 想 实 现 动 态 代 理 机 制 , 需 要java.lang.reflect.InvocationHandler 接 口 和java.lang.reflect.Proxy 类的支持。

运行结果:

代理前 你可以做的事情

我被代理了,哈哈!

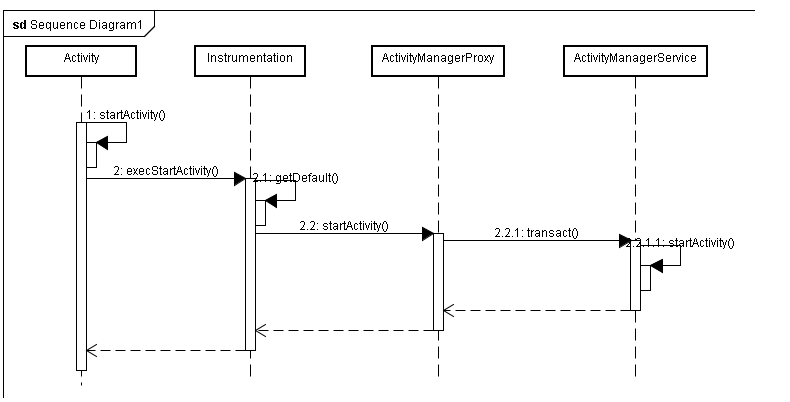
代理后 你可以做的事情

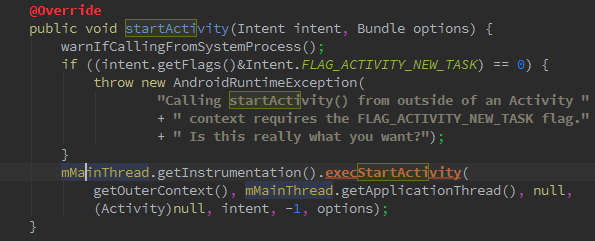
5、分析Android源代码找到 Hook 点

Hook 的理解:

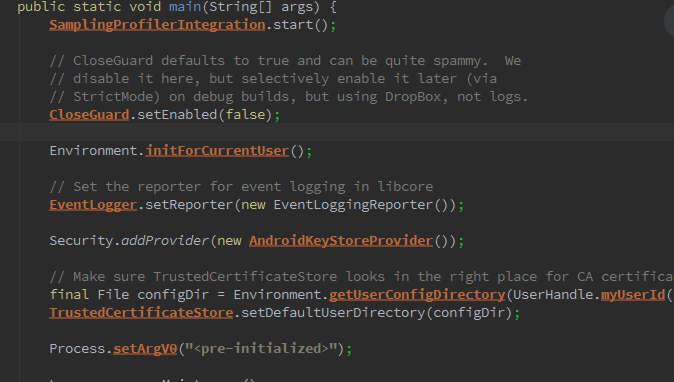
如果我们自己创建代理对象,然后把原始对象替换为我们的代理对象,那么就可以在这个代理对象为所欲为。我们来实现 Hook 掉 startActivity 这个方法,当每次调用这个方法的时候来做我们需要做的事情,这里只打印了一些信息,理解思想。

首先分析我们需要 hook 哪些对象,也就是说我们要 hook 的地方,什么样的对象比较好 Hook 呢?一般是容易找到和不容易改变的对象,这思路就来了,不改变一般在我们类的里面单例对象是唯一对象,静态变量我们一般加上 final static 修饰,有了 final 就不会改变,不变模式里面比如 String 类,里面就很多 final,我们更加这些找到 hook 的地方。我们来看一下 Activity 的 startActivity 方法的时序图:

 从 图 中 分 析 一 下 startActivity 的 调 用 链 , 找 出 合 适 的 Hook 点 。 我 们 知 道 对 于Context.startActivity,由于 Context 的实现实际上是 ContextImpl;我们看 ConetxtImpl 类的startActivity 方法:

 这里,实际上使用了 ActivityThread 类的 mInstrumentation 成员execStartActivity方法;注意到,ActivityThread 实际上是主线程,而主线程一个进程只有一个,因此这里是一个良好的 Hook 点。我们来分析下 ActivityThread 这个类,这个类非常关键,也就是我们常说的主线程、

UI 线程,这个类的 main 函数是程序的入口,如下图：

接 下 来就是 想 要 Hook 掉 我 们的 主线 程 对 象, 也就 是 把这 个 主 线程 对象 里 面 的

mInstrumentation 给替换成我们修改过的代理对象;要替换主线程对象里面的字段,首先我们 得 拿 到 主 线 程 对 象 的 引 用 , 如 何 获 取 呢 ? ActivityThread 类 里 面 有 一 个 静 态 方 法currentActivityThread 可以帮助我们拿到这个对象类;但是 ActivityThread 是一个隐藏类,我们需要用反射去获取,代码如下:

*//获取当前的ActivityThread对象*

Class<?>activityThreadClass = Class.*forName*(***ACTIVIT\_THREAD***);

Method currentActivityThreadMethod = activityThreadClass.getDeclaredMethod(***CURRENT\_ACTIVITY\_THREAD***);

currentActivityThreadMethod.setAccessible(**true**);

Object currentActivityThread = currentActivityThreadMethod.invoke(**null**);

拿到这个 currentActivityThread 之后,我们需要修改它的 mInstrumentation 这个字段为我们 的代 理 对象, 我们 先 实现 这个代 理 对象 ,由于 JDK 动 态代 理 只支持 接口 ,而这 Instrumentation 是一个类,没办法,我们只有手动写静态代理类,覆盖掉原始的方法即可。

//这个方法是由于原始方法里面的Instrumentation有execStartActivity方法来定的

public ActivityResult execStartActivity(Context who, IBinder contextThread, IBinder token, Activity target,

Intent intent, int requestCode, Bundle options) {

Log.d(TAG, "\n打印调用startActivity相关参数: \n" + "who = [" + who + "], " +

"\ncontextThread = [" + contextThread + "], \ntoken = [" + token + "], " +

"\ntarget = [" + target + "], \nintent = [" + intent +

"], \nrequestCode = [" + requestCode + "], \noptions = [" + options + "]");

Log.i(TAG, "------------hook success------------->");

Log.i(TAG, "这里可以做你在打开StartActivity方法之前的事情");

Log.i(TAG, "------------hook success------------->");

Log.i(TAG, "");

//由于这个方法是隐藏的，所以需要反射来调用，先找到这方法

try {

Method execStartActivity = Instrumentation.class.getDeclaredMethod(

EXEC\_START\_ACTIVITY,

Context.class, IBinder.class, IBinder.class, Activity.class,

Intent.class, int.class, Bundle.class);

execStartActivity.setAccessible(true);

return (ActivityResult) execStartActivity.invoke(oldInstrumentation, who,

contextThread, token, target, intent, requestCode, options);

} catch (Exception e) {

//如果你在这个类的成员变量Instrumentation的实例写错mInstrument,代码讲会执行到这里来

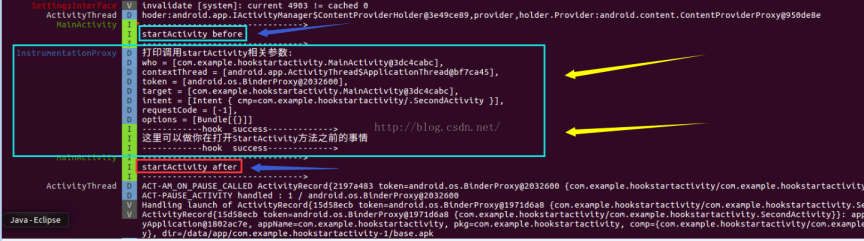
throw new RuntimeException("if Instrumentation paramerter is mInstrumentation, hook will fail");

}

}

Ok,有了代理对象,我们要做的就是偷梁换柱!代码比较简单,采用反射直接修改:

通过pidcat.py工具运行结果：



修改了一下代码：

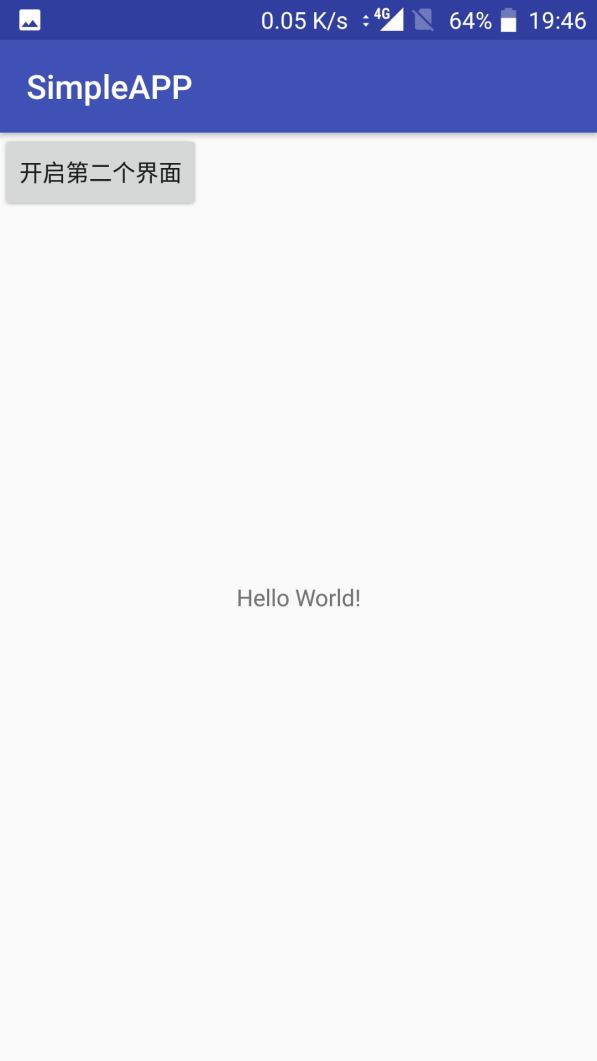
将InstrumentionProxy修改了下，在应用startActivity时添加了对话框选项，如下图所示：



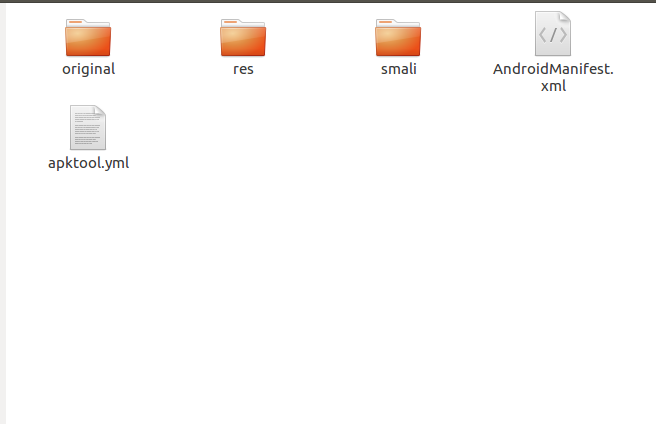
6. 将Hook代码注入另外一个APK中

由上文所知我们是编写在自己的应用里，所以不会出现很多问题，也只需要理解就可以了。但是我们的最终目标将代码注入一个现成的APK中，然后Hook它。

首先我们先自己开发个很简单的应用，并且要加签名，应用执行如下图所示：

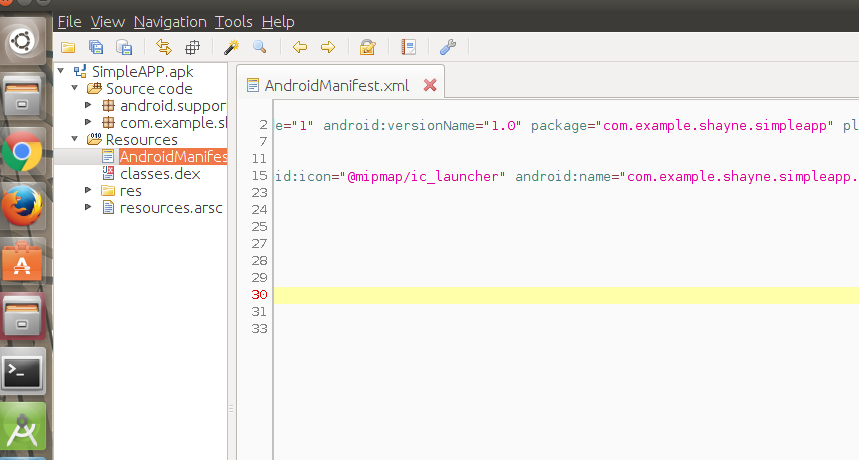


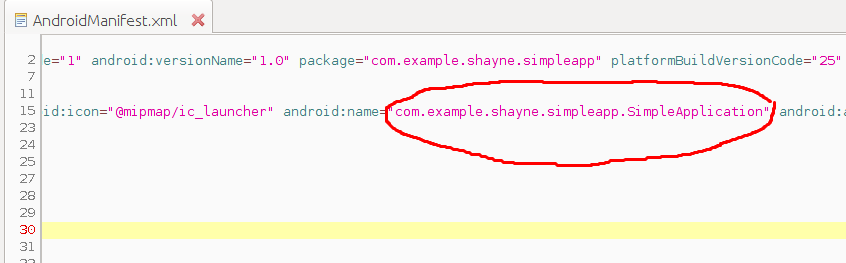
通过apktool进行对刚才的应用进行解包 apktool d 路径名+SimpleAPP.apk，如下图所示：



用jadx-gui进行反编译，对现成的SimpleAPP.APK进行分析，执行jadx-gui SimpleAPP.apk

如下图所示：





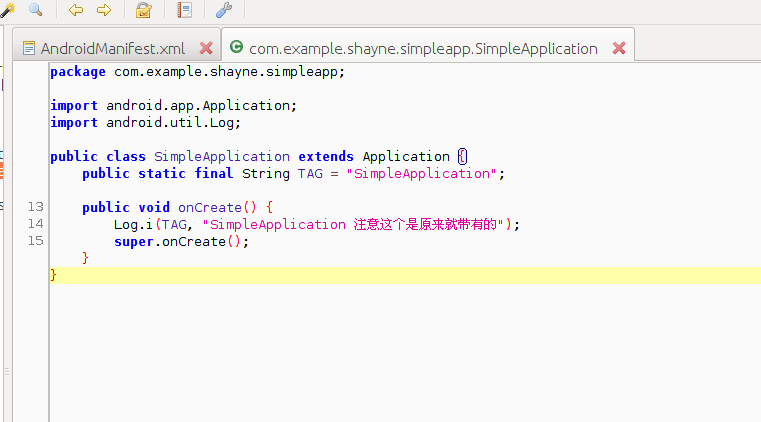
如上图可知，此APK有属于自己的Application，一般在我们的市场中，大部分的APK都会有属于自己的Application，假设没有属于自己的Application的话，我们只需要在manifest里面的添加android:name="com.example.shayne.hookactivity.MyApplication"这段话，因为我们 的代码必须要在应用中运行，所以我们必须先初始化对象，并且在应用已开启的时候就运行我们自己的代码。如下图所示：



但是在生活中我们平时用的APK都是有自己Application，然而上面的实现方式是不可行，如果硬要这么做，只能替换它，然后运行自己代码后必须要恢复它的Application里的代码，这样反而特别的麻烦。所以简单的做就是不改manifest里的Application，把他Application里的最终的父类设置成自己的Application。

步骤如下：

1.先在jadx-gui反编译apk里查看Application。

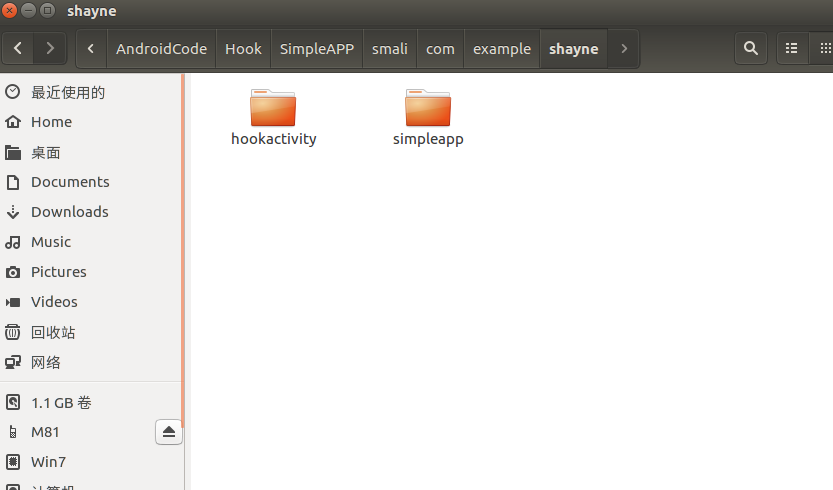


2.如上图可知：SimpleApp.apk的SimpleApplication直接继承系统所带的Application，所以我们只需将Application替换就可以了。注意这里替换要在Apktool解包后的源文件里修改，修改里面的smali代码。找到SimpleApplication.smali文件。然后修改里面的内容。将Landroid/app/Application改成Lcom/example/shayne/hookactivity/MyApplication。如下图所示：



3.如上图可知，只需要修改两个地方就可以了。第一个地方是SimpleApplication所继承的父类从系统 的Application改成自己的MyApplication。第二个地方是将SimpleApplication里的onCreate中的super.onCreate，改成调用自己MyApplication里的onCreate。

4.然后apktool d 路径+HookActivity.apk，将HookActivity的smali代码拷贝到解包后的HookAPP里。注意这里的路径名要对，如下图所示：

 5.然后Apktool b 路径名+HookAPP文件夹。这样二次打包APK了。但是还需要重新签名。

6.在这里 用的是Auto-sign自动签名工具，签名后安装在手机上就可以了。效果图如下：



上面的开发中只是自己个人的Apk进行Hook，本人也试了下别的应用如WPS，但是在代码注入别的应用的时候要注意人家的Activity的开启，一般的应用第一个Activity不会有界面，只是做一些开启的作用，这时候在这个Activity开启一个对话框会出现严重的错误，需要查看log，因为这个Activity没有Window导致系统不能在这里面进行绘制对话框，所以引发错误，这里就需要另外一种方法，就是开启一个自己的Activity并且有界面。具体还需要更深入研究。效果图如下：



总结：

整体来讲这里是有缺陷的，比如代码注入的时候是很麻烦的。但是其实也不麻烦，可以写个脚本，将上面的操作步骤进行自动化处理就可以了。如果HookActivity的代码能够功能更强大点，能兼容所以的App，这样就可以所有App进行代码注入，然后Hook这个App就可以实现很多功能了。这个还有待开发。

从上面的Hook startActivity来说，替换了系统的 Instrumentation,理解了 hook 思想。也让我们为插件化开发更加的深入了一步。在现实生活中，很多App已经实现了插件化开发，如：淘宝，还有微信等等。

插件化开发参考博客：http://blog.csdn.net/singwhatiwanna/article/details/39937639