"相传江湖上有种武功亦正亦邪, 灵活多变, 正邪两派都在其基本心法的基础上将其发扬光大, 开创了很多独步武林的武功秘籍"

0x0 概述

HOOK(钩子)技术能再事件传递过程中截获, 修改, 监控事件信息, 就像一个钩子可以挂事件上面, 江湖人称催眠术, 能读取改变一个人的记忆,
是不是想想就可怕, 正道引导迷途的人类走向光明, 邪道引诱人堕落进入深渊HOOK就像精神控制大师饲养的大脑记忆区域的寄生虫, 可以挂
接到记忆神经元上面读取某段记忆, 也能在这串记忆流上进行拦截篡改, 甚至使你产生幻觉。然后达到监控、篡改某个人(进程)记忆(数据)的目的。

对于邪道, 他们使用这些技术

1. 制作外挂程序, HOOK程序, 篡改程序数据, 制作外挂, 如游戏外挂, 改机类软件
2. 制作病毒木马程序, HOOK操作系统相关进程, 监控被感染者的电脑, HOOK杀毒软件, 绕过杀毒软件检测
3. HOOK某些程序的机密数据, 获取明文数据

对于正道, 他们使用这些技术

1. 制作沙盒类程序, 监控程序的运行
2. 杀毒软件HOOK操作系统API, 拦截恶意病毒木马行为
3. 某些操作系统的某些消息机制, 提供给软件开发者丰富的功能接口
4. 程序自身HOOK操作系统的某些功能, 来丰富操作系统提供的功能

有江湖的地方就有门派, 各大门派根据HOOK的原理, 创造了很多武林秘籍(HOOK框架), 从而开宗立派, 传为江湖一段佳话

0x1 xpose门派简介

当年rovo89老祖在无数个夜晚苦思冥想, 终于感悟天道, 创下这独步武林的xpose大法, 从此开宗立派, 成为Android这一方小世界的精神控制的第一大派.
无数江湖人士拜其门下, 苦习这xpose大法. 新手若习得这xpose, 不出数日, 便可功至化境, 轻易便可操作普通人的思维意识,
高手若习得这xpose大法变可神出鬼没, 藏匿于意海丹田, 所控之人秋毫无知

江湖人士若练此功, 需到此处配置丹药(**搭建环境**), 洗精伐髓, 脱胎换骨
http://repo.xposed.info/module/de.robv.android.xposed.installer

江湖人士在洗精伐髓之后需按下面功法研习(**API文档**), 早日打通任督, 以功法练制独特的记忆寄生虫
http://api.xposed.info/reference/de/robv/android/xposed/IXposedHookInitPackageResources.html

亦有江湖中人, 设立演武堂(**论坛**), 交流功法心得, 门庭若市, 好不热闹
https://forum.xda-developers.com/xposed

更有门派成员, 将其辛苦所练制的各种功能的记忆寄生虫, 寄于市场(**xpose各种插件市场**)供人使用
http://repo.xposed.info/module-overview

rovo89老祖其人心胸开阔, 将武功心法尽数公开, 放到**github**藏经阁地址: https://github.com/rovo89/XposedInstaller, 供给世人参考改良

古人云: 有此欣欣向荣之态, 岂能不壮哉!

0x2 xpose入门心得

"习武之人都知道, 武功由浅入深, 先练其形, 在练其意, 意形结合, 方至大成"

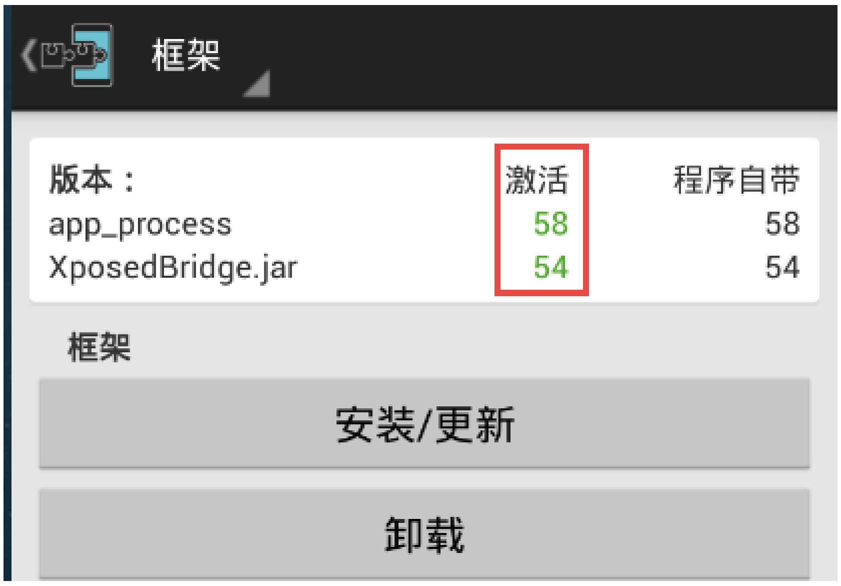
1.下载相关工具

XposedInstaller下载
http://repo.xposed.info/module/de.robv.android.xposed.installer

XposedBridged.jar下载
https://github.com/rovo89/XposedBridge/releases
http://forum.xda-developers.com/xposed/xposed-api-changelog-developer-news-t2714067

2.安装XposedInstaller并激活

首先我们需要安装Xpose
激活步骤: 启动XposedInstaller -> 框架 -> 安装更新 ->模拟器重启 (ps:模拟器会直接屏幕黑掉,直接结束进程即可,不行就反复试几下)
最好是点击软重启
激活后这里会有绿色的数字信息

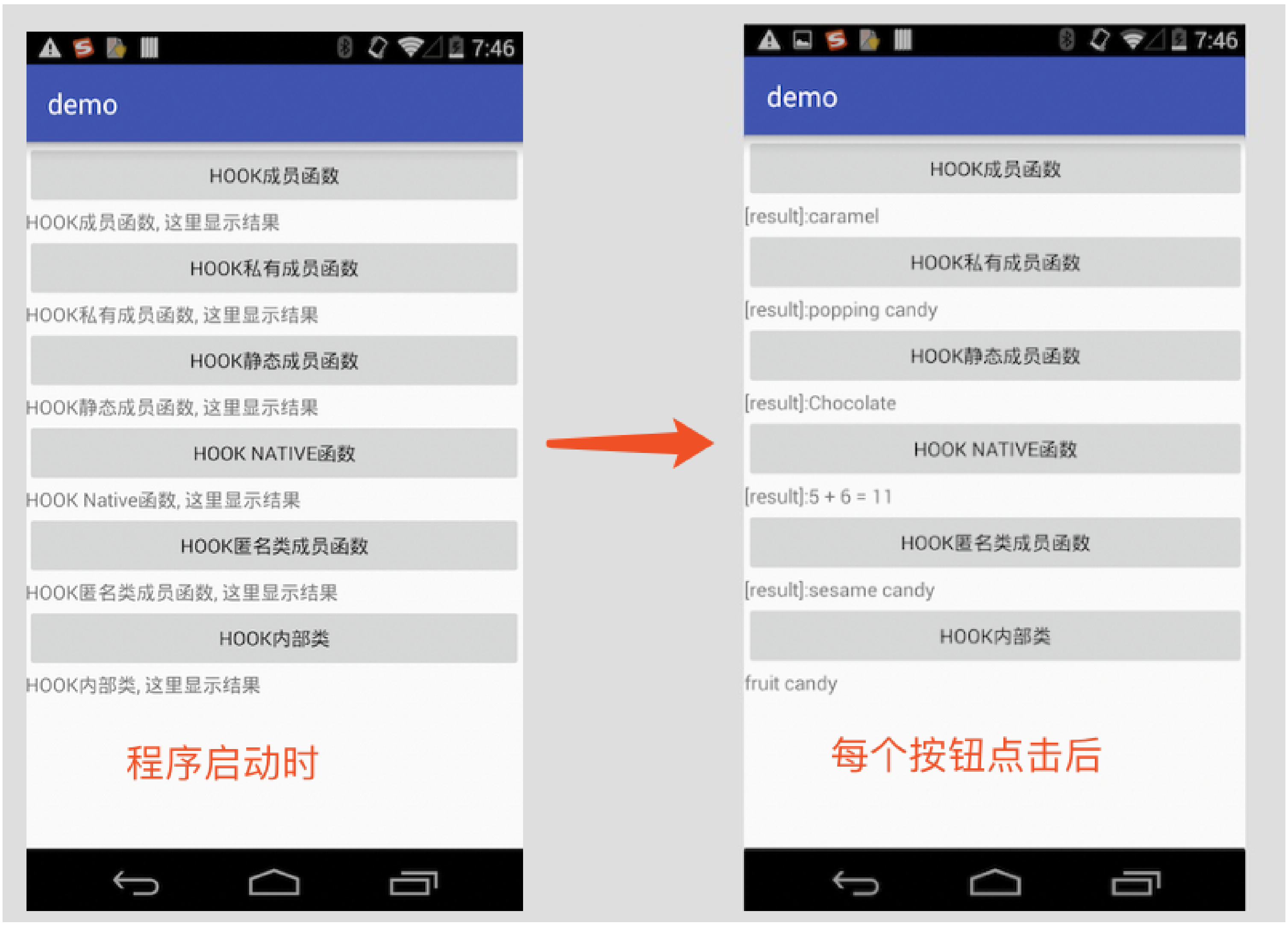


版本：	激活	程序自带
app_process	58	58
XposedBridge.jar	54	54

框架

3.Android Studio新建一个测试工程(被Hook的APP)

测试工程名称: demo
程序运行后效果如下:
每个按钮调用对应语法的函数, 并呈现到对应的TextView上



Hook的类简单的展示了几种基本语法函数, 跟多代码请参考附录后面的源码

```
public class Candy {
    public String mProperty = "default";

    // Used to load the 'native-lib' library on application startup.
    static {
        System.loadLibrary("native-lib");
    }

    public Candy(String property) {
        /* 带参构造 */
        mProperty = property;
    }

    /* 成员函数 */
    public String Caramel(String incantation) {
        return incantation;
    }

    /* 私有成员函数 */
    private String PoppingCandy(String incantation) {
        return incantation;
    }
    public String callPoppingCandy(String incantation) {
        return PoppingCandy(incantation);
    }
    /* 静态成员函数 */
    public static String ChocolateCandy(String incantation) {
        return incantation;
    }

    /* Native函数 */
    public static native String CottonCandy(int a, int b);

    public class InteriorCandy {

        /* 内部类成员函数 */
        public String FruitCandy(String incantation) {
            return incantation;
        }
    }
}
```

4.新建我们的XposedHook工程

●在AndroidManifest文件中加入如下代码 放在Application中

```
<meta-data
   android:name="xposedmodule"
   android:value="true" />
<meta-data
   android:name="xposeddescription"
   android:value="Easy example" />              <!--  这里是关于插件的说明，会显示到Xpose插件上  -->
<meta-data
   android:name="xposedminversion"
   android:value="54" />
```

●新建lib目录

PS: 必须是lib目录，不能是libs，否则会报错:

Class ref in pre-verified class resolved to unexpected implementation

将下载好的XposedBridged.jar放入该目录

并右键->Add To Library   这个步骤会在grandlew中添加

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   testCompile 'junit:junit:4.12'
   compile 'com.android.support:appcompat-v7:23.1.1'
   compile files('lib/XposedBridgeApi-54.jar')
}
```

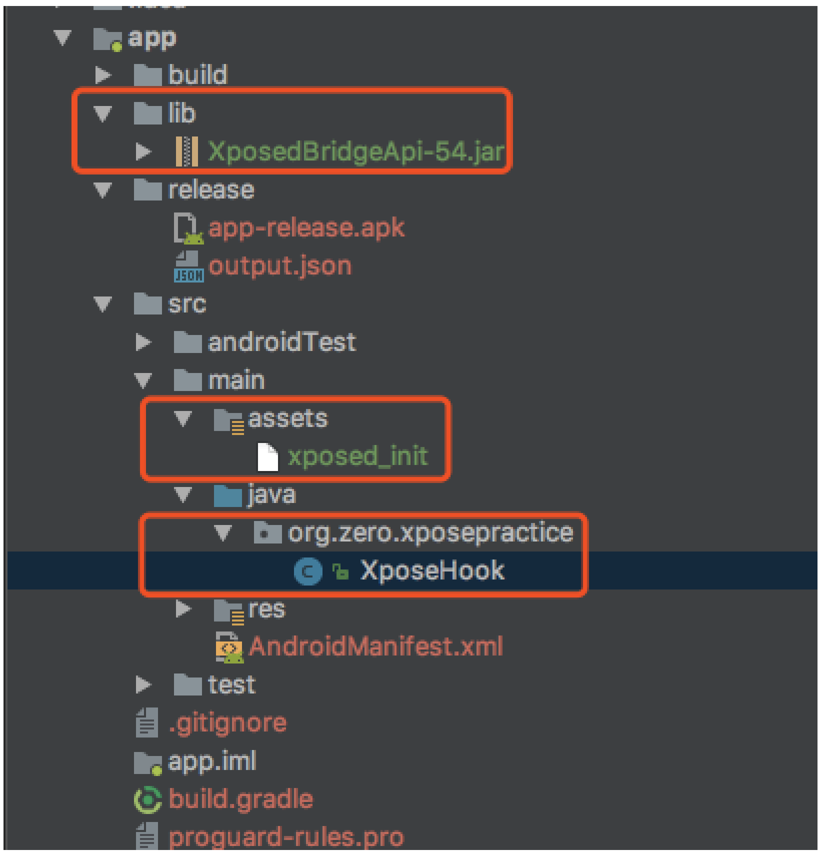
我们要将`compile files`修改为`provided files`,最后效果如下，新版本Android变成`implementation`我没有试过implementation有没有用，有兴趣的可以试试

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
```

```
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    provided files('lib/XposedBridgeApi-54.jar')
}
```

●添加assets目录
在该目录下添加xposed_init
该文件的作用是指定module入口类,Hook的实现代码在该类中
格式: 包名称 + 类名
org.zero.xposepractice.XposeHook

●工程结构
此时工程的目录结构是这个样子的
PS: 最好先别把Activity删掉, 不然每次都要打release包进行测试



```
▼ app
  ▶ build
  ▼ lib
    ▶ XposedBridgeApi-54.jar
  ▼ release
      app-release.apk
      output.json
  ▼ src
    ▶ androidTest
    ▼ main
      ▼ assets
          xposed_init
      ▼ java
        ▼ org.zero.xposepractice
            XposeHook
      ▶ res
        AndroidManifest.xml
    ▶ test
    .gitignore
    app.iml
    build.gradle
    proguard-rules.pro
```

●新建xposed_init中指明的入口类XposedHook

```
package org.zero.xposepractice;

import android.util.Log;

import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XposedBridge;
import de.robv.android.xposed.XposedHelpers;
import de.robv.android.xposed.callbacks.XC_LoadPackage;

/**
 * Created by bingghost on 2017/12/7.
 *      xpose hook demo工程
```

```java
 */
public class XposeHook implements IXposedHookLoadPackage {
    public static final String TAG_HOOK = "__BING_HOOK";

    /* 配置HOOK的包名 */
    private static final String TARGET_PACKAGE_NAME = "org.zero.demo";

    private boolean isTargetPackage(String currentPackage, String targetPackage) {
        return currentPackage.equals(targetPackage);
    }

    @Override
    public void handleLoadPackage(XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable {
        /* 如果载入的不是指定包名就退出 */
        if (!isTargetPackage(loadPackageParam.packageName, TARGET_PACKAGE_NAME)) {
            return;
        }
        XposedBridge.log("Loaded app: " + loadPackageParam.packageName);
        Log.v(TAG_HOOK, "Hook Demo Load success!!!");

        /* hook 成员函数*/
        demoHookMemberFunction(loadPackageParam);

        /* hook 私有成员函数*/
        demoHookPrivateMemberFunction(loadPackageParam);

        /* hook 静态函数*/
        demoHookStaticFunction(loadPackageParam);

        /* hook native函数 */
        demoHookNativeFunction(loadPackageParam);

        /* hook 匿名函数 */
        demoHookAnonymousFunction(loadPackageParam);

        /* Hook 内部类成员函数 */
        demoHookInnerClassMemberFunction(loadPackageParam);
    }

    /* Hook 内部类成员函数 */
    private void demoHookInnerClassMemberFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
        XposedHelpers.findAndHookMethod("org.zero.demo.Candy$InteriorCandy",
                loadPackageParam.classLoader,
                "FruitCandy",
                String.class,
                new XC_MethodHook() {
                    protected void afterHookedMethod(MethodHookParam param) {

                    }

                    protected void beforeHookedMethod(MethodHookParam param) {
                        /* 修改参数 */
                        param.args[0] = "(Hook FruitCandy)";
                    }
                });
    }

    /* hook 匿名函数 */
    private void demoHookAnonymousFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
        XposedHelpers.findAndHookMethod("org.zero.demo.MainActivity$5$1",
                loadPackageParam.classLoader,
                "Candy",
                String.class,
                new XC_MethodHook() {
                    protected void afterHookedMethod(MethodHookParam param) {

                    }

                    protected void beforeHookedMethod(MethodHookParam param) {
                        /* 修改参数 */
                        param.args[0] = "(Hook Annotation Candy)";
                    }
                });
    }

    /* hook native函数 */
    private void demoHookNativeFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
        XposedHelpers.findAndHookMethod("org.zero.demo.Candy",
                loadPackageParam.classLoader,
                "CottonCandy",
                int.class,
                int.class,
                new XC_MethodHook() {
                    protected void afterHookedMethod(MethodHookParam param) {
                        Integer  para1 =  (Integer) param.args[0];   // 获取参数1
                        Integer para2 = (Integer) param.args[1];     // 获取参数2
                        String s1 = Integer.toString(para1);
                        String s2 = Integer.toString(para2);
                        Log.v(TAG_HOOK,"hook param1:" + s1);
                        Log.v(TAG_HOOK,"hook param2:" + s2);
                    }

                    protected void beforeHookedMethod(MethodHookParam param) {
                        param.args[0] = 10;
                        param.args[1] = 14;
                    }
                });
    }

    /* hook 静态函数*/
    private void demoHookStaticFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
        XposedHelpers.findAndHookMethod("org.zero.demo.Candy",
                loadPackageParam.classLoader,
                "ChocolateCandy",
                String.class,
                new XC_MethodHook() {
                    protected void afterHookedMethod(MethodHookParam param) {

                    }

                    protected void beforeHookedMethod(MethodHookParam param) {
                        /* 修改参数 */
                        param.args[0] = "(Hook ChocolateCandy)";
                    }
                });
    }
    /* hook 私有成员函数*/
    private void demoHookPrivateMemberFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
        XposedHelpers.findAndHookMethod("org.zero.demo.Candy",
                loadPackageParam.classLoader,
                "PoppingCandy",
                String.class,
                new XC_MethodHook() {
                    protected void afterHookedMethod(MethodHookParam param) {

                    }

                    protected void beforeHookedMethod(MethodHookParam param) {
                        /* 修改参数 */
                        param.args[0] = "(Hook PoppingCandy)";
                    }
                });
    }

    /* hook 成员函数*/
    private void demoHookMemberFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
        XposedHelpers.findAndHookMethod("org.zero.demo.Candy",
                loadPackageParam.classLoader,
                "Caramel",
                String.class,
                new XC_MethodHook() {


                    protected void afterHookedMethod(MethodHookParam param) {
                        /* 修改结果 */
                        String str = (String) param.getResult();
                        Log.v(TAG_HOOK,"hook after result :" + str);

                        String  arg1 =  (String) param.args[0];      // 获取参数1
                        param.setResult("(Hook Caramel)");           // 设置返回值

                        Log.v("get Caramel param1:", arg1);
                    }

                    protected void beforeHookedMethod(MethodHookParam param) {

                    }
                });
    }
}
```

相关的函数说明:

handleLoadPackage	包加载时会调用
afterHookedMethod	Hook函数调用后, 一般作为hook函数执行结果的时机
beforeHookedMethod	Hook函数调用前, 一般作为hook参数的时机
XposedBridge.log	打印的内容将在XposedInstall的日志界面

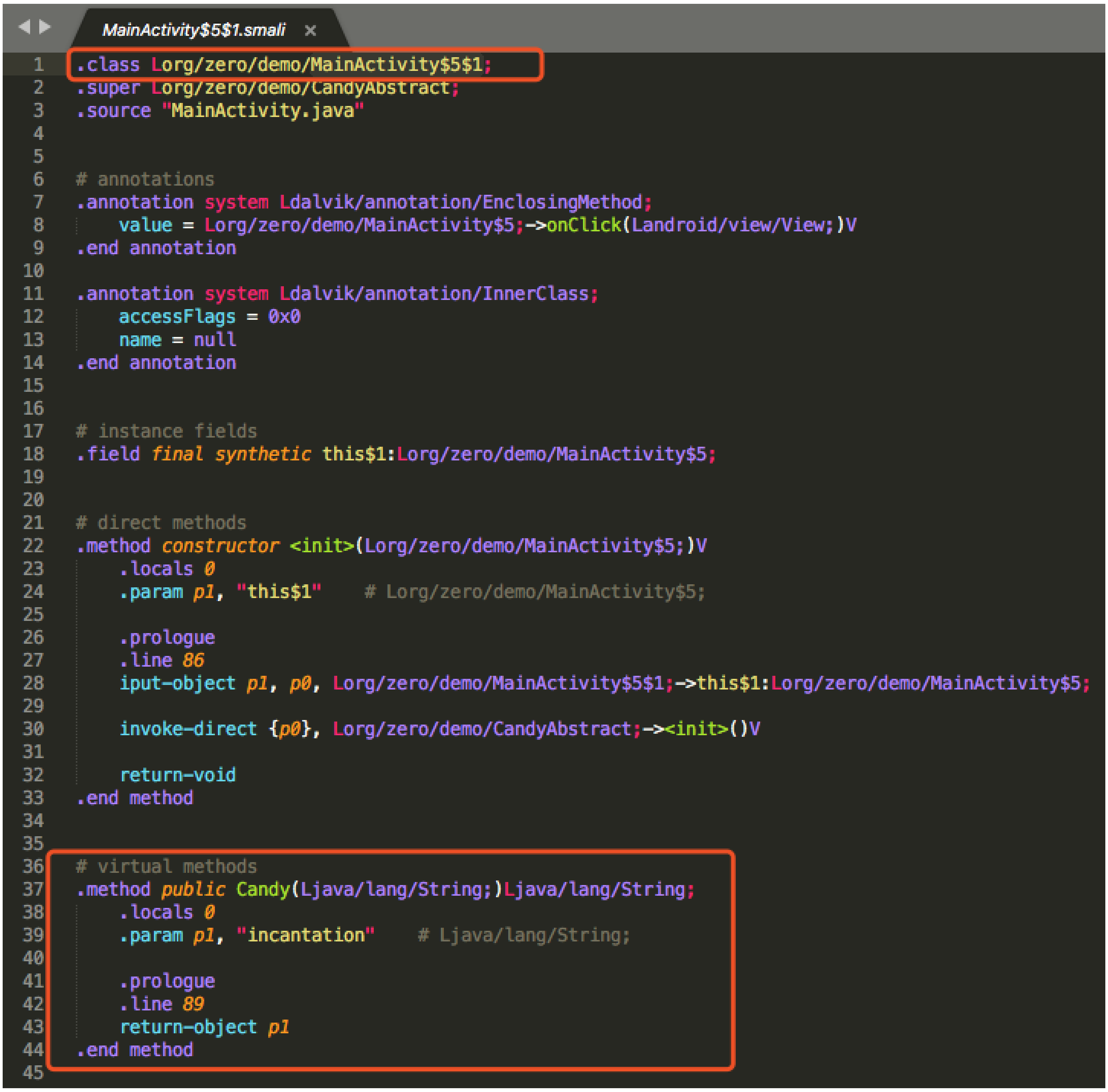
匿名类和内部类的HOOK说明

那么此时一定会有疑问, 匿名类和内部类如何确定hook函数的

(1). 对于匿名类源代码如下:

```java
/* 匿名类成员函数 */
Button button5 = findViewById(R.id.button5);
button5.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        CandyAbstract candy = new CandyAbstract() {
            @Override
            public String Candy(String incantation) {
                return incantation;
            }
        };

        String result = candy.Candy("sesame candy");
        mTextView5.setText("[result]:" + result);
    }
});
```

我们用apktool对包进行解包, 发现匿名类变成:



```smali
.class Lorg/zero/demo/MainActivity$5$1;
.super Lorg/zero/demo/CandyAbstract;
.source "MainActivity.java"


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lorg/zero/demo/MainActivity$5;->onClick(Landroid/view/View;)V
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation


# instance fields
.field final synthetic this$1:Lorg/zero/demo/MainActivity$5;


# direct methods
.method constructor <init>(Lorg/zero/demo/MainActivity$5;)V
    .locals 0
    .param p1, "this$1"    # Lorg/zero/demo/MainActivity$5;

    .prologue
    .line 86
    iput-object p1, p0, Lorg/zero/demo/MainActivity$5$1;->this$1:Lorg/zero/demo/MainActivity$5;

    invoke-direct {p0}, Lorg/zero/demo/CandyAbstract;-><init>()V

    return-void
.end method


# virtual methods
.method public Candy(Ljava/lang/String;)Ljava/lang/String;
    .locals 0
    .param p1, "incantation"    # Ljava/lang/String;

    .prologue
    .line 89
    return-object p1
.end method
```

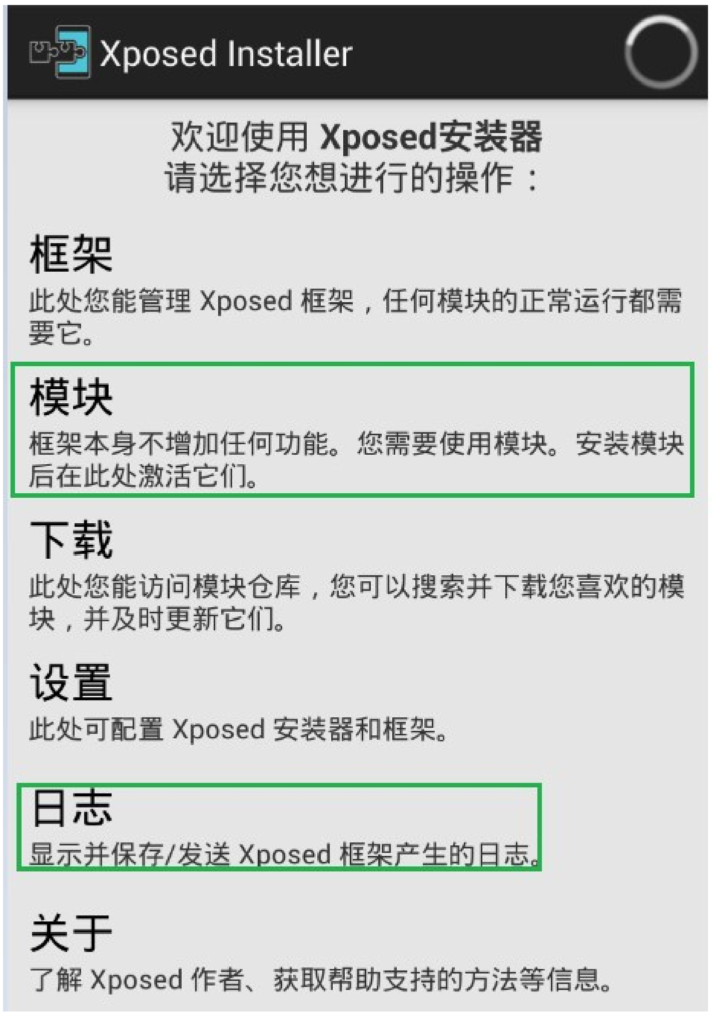
所以实际分析的时候, 以smali文件中.class的类名称为主就行了, 所以对于本例的匿名类hook代码有:

```
/* hook 匿名函数 */
private void demoHookAnonymousFunction(XC_LoadPackage.LoadPackageParam loadPackageParam) {
    XposedHelpers.findAndHookMethod("org.zero.demo.MainActivity$5$1",
            loadPackageParam.classLoader,
            "Candy",
            String.class,
            new XC_MethodHook() {
                protected void afterHookedMethod(MethodHookParam param) {

                }

                protected void beforeHookedMethod(MethodHookParam param) {
                    /* 修改参数 */
                    param.args[0] = "(Hook Annotation Candy)";
                }
            });
}
```

同理, 内部类函数都是按照这个方法去找, 当然对于抽象函数得找到具体的实现类

●安装运行我们的xpose插件

安装好XposedDemoAPP 在模块中勾选上重启系统 然后软重启



Xposed Installer

欢迎使用 Xposed安装器
请选择您想进行的操作：

框架
此处您能管理 Xposed 框架，任何模块的正常运行都需要它。

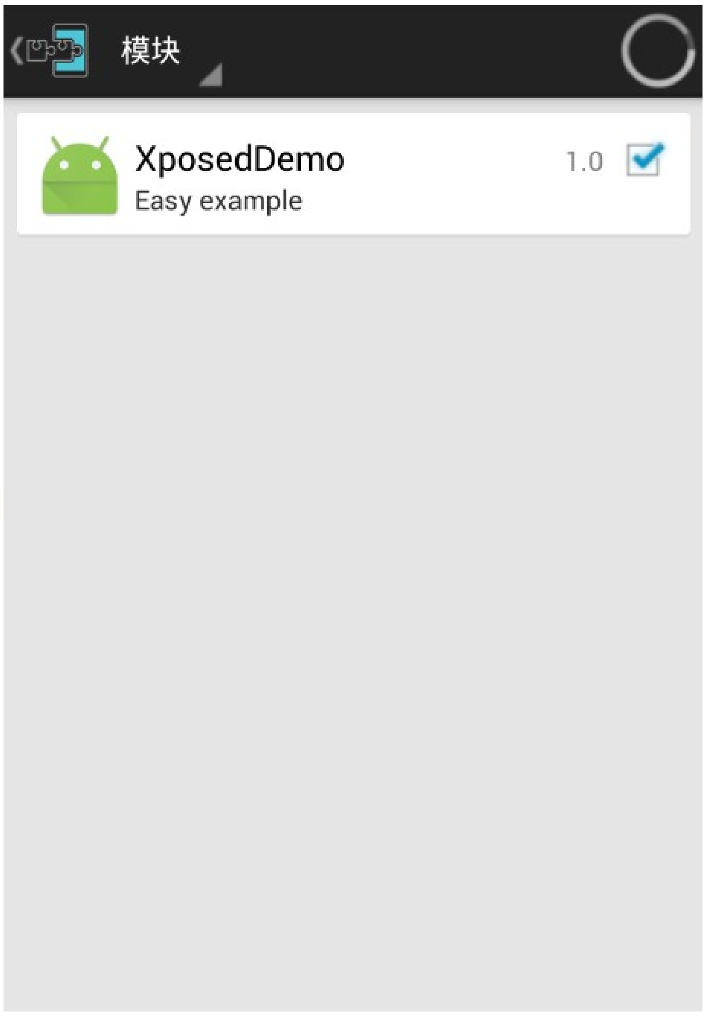
模块
框架本身不增加任何功能。您需要使用模块。安装模块后在此处激活它们。

下载
此处您能访问模块仓库，您可以搜索并下载您喜欢的模块，并及时更新它们。

设置
此处可配置 Xposed 安装器和框架。

日志
显示并保存/发送 Xposed 框架产生的日志。

关于
了解 Xposed 作者、获取帮助支持的方法等信息。

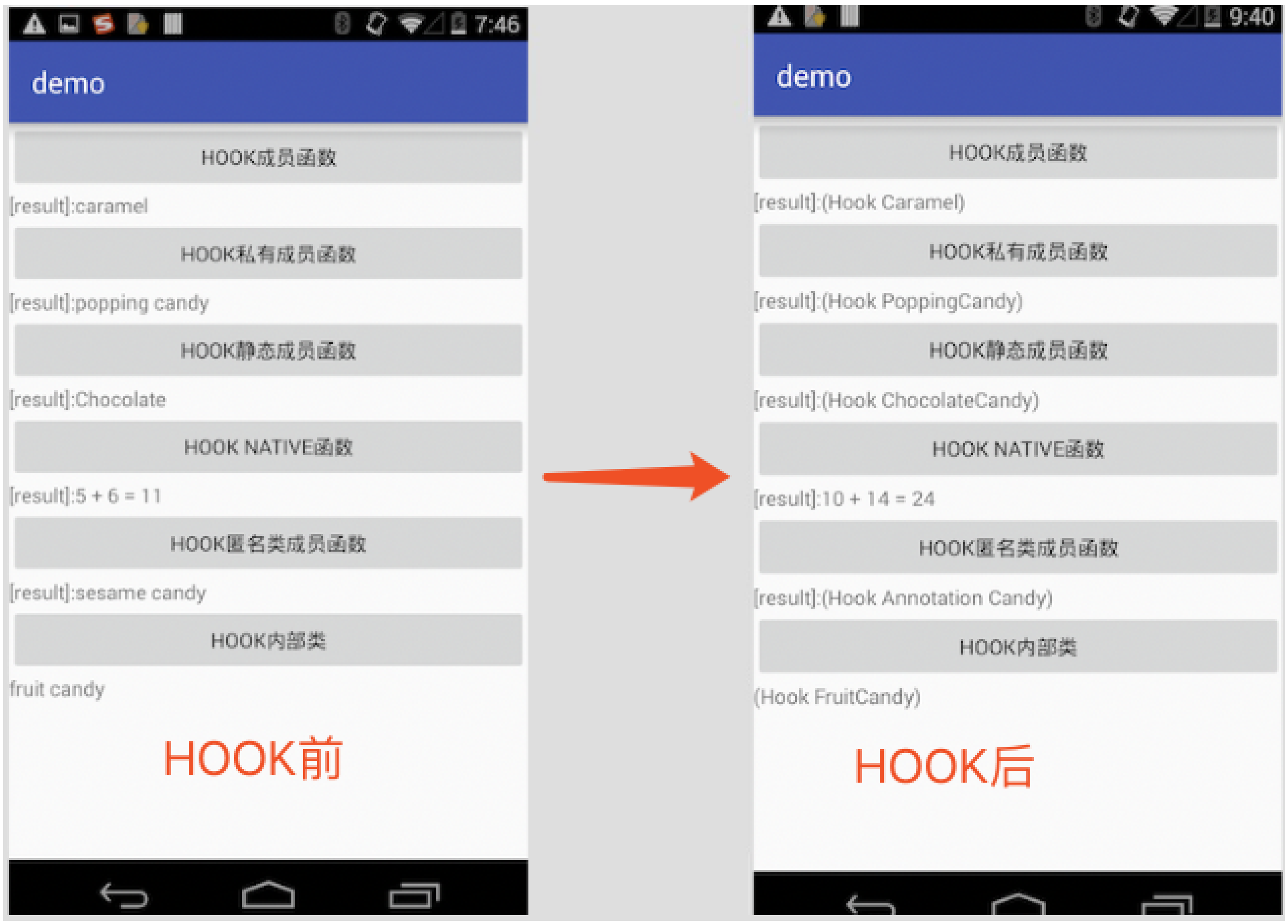


模块

XposedDemo 1.0 ✓
Easy example

5.运行结果

测试APP显示结果如下:



```
[result]:caramel
[result]:popping candy
[result]:Chocolate
[result]:5 + 6 = 11
[result]:sesame candy
fruit candy
```

```
[result]:(Hook Caramel)
[result]:(Hook PoppingCandy)
[result]:(Hook ChocolateCandy)
[result]:10 + 14 = 24
[result]:(Hook Annotation Candy)
(Hook FruitCandy)
```

0x3 总结

本篇主要讲述了hook的基本概念，以及xpose的基本使用方法和注意事项，避免在使用过程中的坑点，这个hook系列我也不知道要写多少篇，看着写吧，尽力写成全网最全最详细的一个系列

附录

本篇幅源码请移步(part1部分):
https://github.com/ZeroPractice/AndroidHookPractice

那么客官，欲知后事如何，且听下回分解