Xposed 框架之函数 Hook 学习

作者：Fly2015

Xposed是Android下Java层的开源Hook框架类似的有cydiasubstrate框架并且据说cydiasubstrate框架能实现 Android 的 Java 层和 Native 层的函数的 Hook。学习 Android 的逆向也有一段时间了，记录一下学习 Xposed 框架的过程。

Xposed 框架的学习网站：http://repo.xposed.info/

一、Xposed 框架实现 Hook 的原理介绍

Zygote 是 Android 的核心，每运行一个 app，Zygote 就会 fork 一个虚拟机实例来运行 app，Xposed Framework 深入到了 Android 核心机制中，通过改造 Zygote 来实现一些很牛逼的功能。Zygote 的启动配置在 /init.rc 脚本中，由系统启动的时候开启此进程，对应的执行文件是 /system/bin/app_process，这个文件完成类库加载及一些函数调用的工作。

当系统中安装了 Xposed Framework 之后，会对 app_process 进行扩展，也就是说，Xposed Framework 会拿自己实现的 app_process 覆盖掉 Android 原生提供的 app_process 文件，当系统启动的时候，就会加载由 Xposed Framework 替换过的进程文件，并且，Xposed Framework 还定义了一个 jar 包，系统启动的时候，也会加载这个包：

/data/data/de.robv.android.xposed.installer/bin/XposedBridge.jar

二、Xposed 框架运行的条件

1.Rooted Device / Emulator （已 root 的手机或者模拟器）

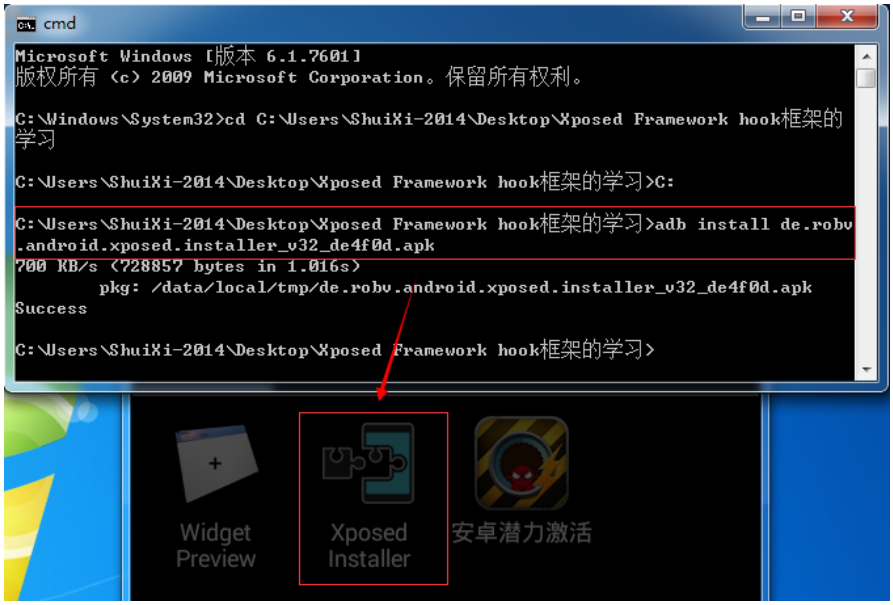
2.Xposed Installer (Xposed 安装程序下载)

3.Hooking Android App （要被 Hook 的目标 App）

Xposed Framework 就是一个 apk 包也就是上面下载的 Xposed 安装程序，下载后用下面的命令安装到手机上或者模拟器：

```
adb install de.robv.android.xposed.installer_v32_de4f0d.apk
```

安装好之后，打开 Xposed ，下面是截图：



三、Hook 模块的实现

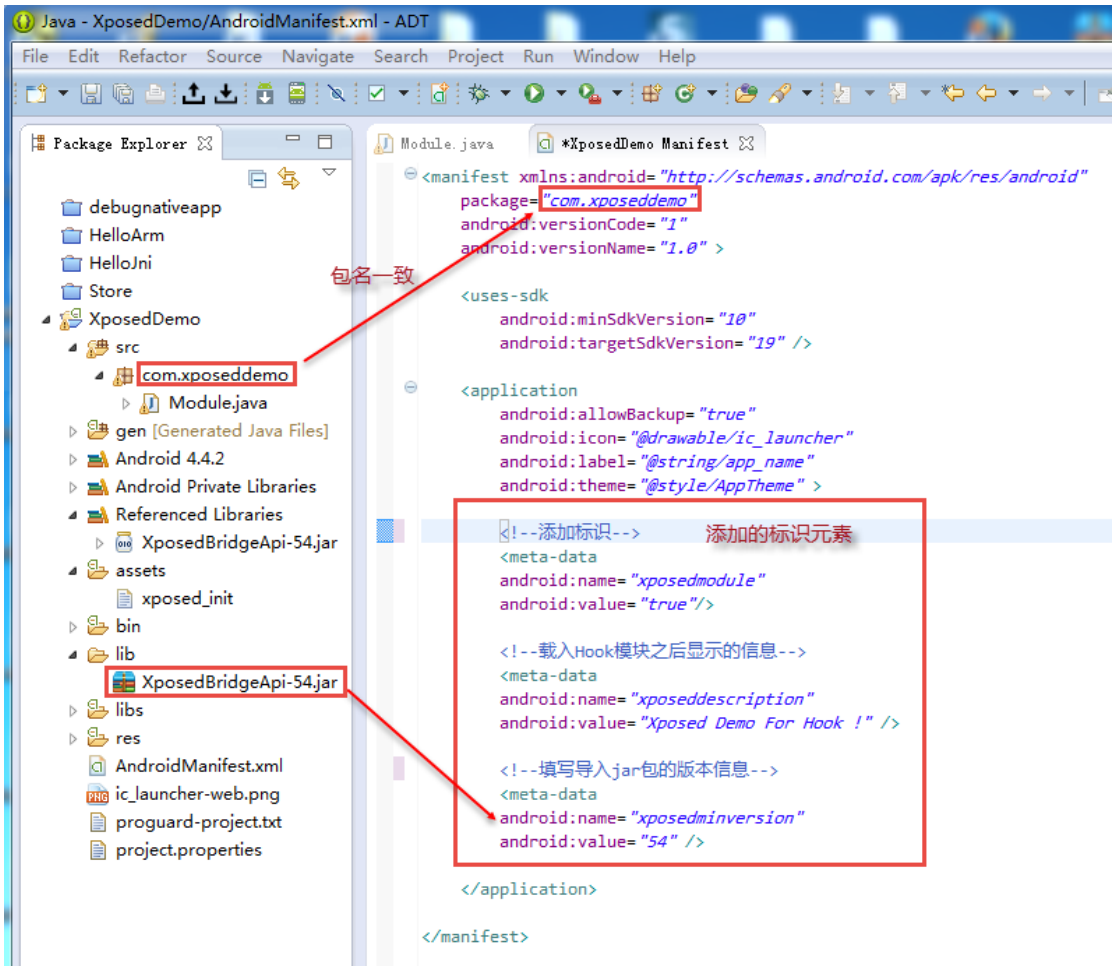
1.添加 meta-data 元素

一个 Xposed 模块就是一个 Android app 不需要实现 Activity 本例中的 Hook 模块叫 com.xposeddemo，下面是这个实例的 AndroidManifest.xml 文件，注意其中定义了三项 meta-data 元素：

1).*xposedmodule*

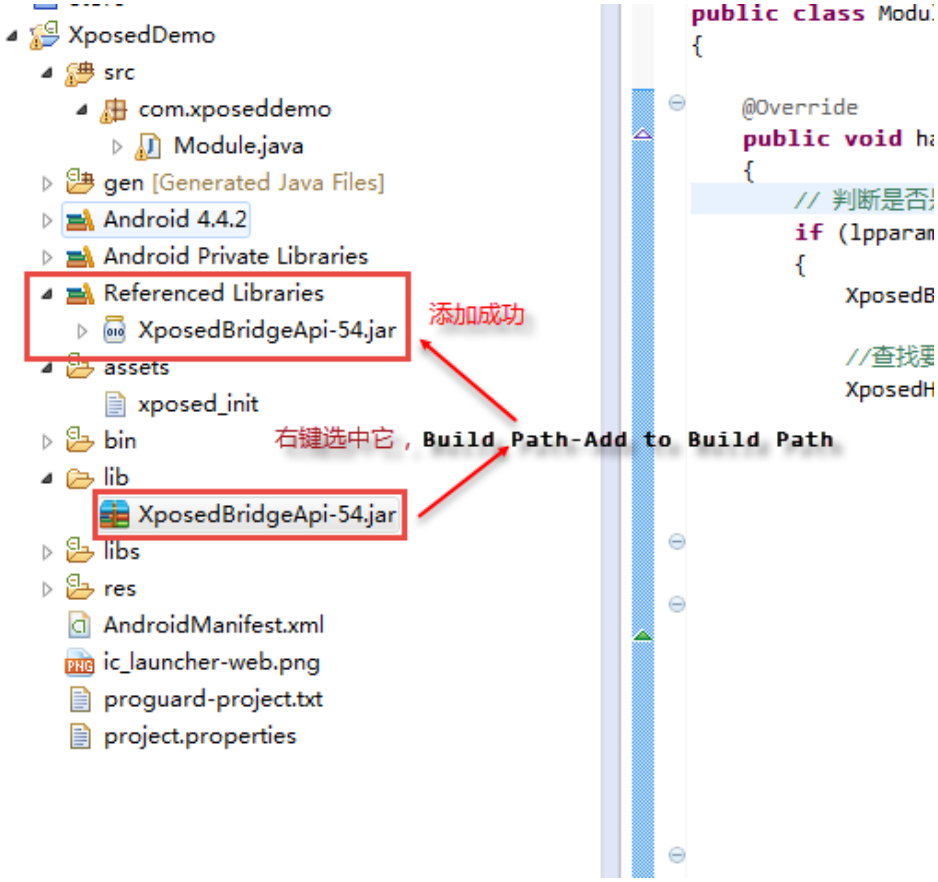
2).*xposeddescription* 载入 Hook 模块之后显示的信息

3).*xposedminversion* 填写导入的 jar 包的版本号



2.添加 XposedBridgeApi-54.jar 包

其中 xposedminversion 是指 XposedBridge library 的版本号信息，需要将 XposedBridge library 复制到 lib 目录(注意是 lib 目录不是 libs 目录)，如果**没有 lib 目录**，在工程目录下新建一个 lib 文件夹，将下载好的 XposedBridgeApi-54.jar 包放入其中，然后在工程里 **右键–Build Path–Add to Build Path.**



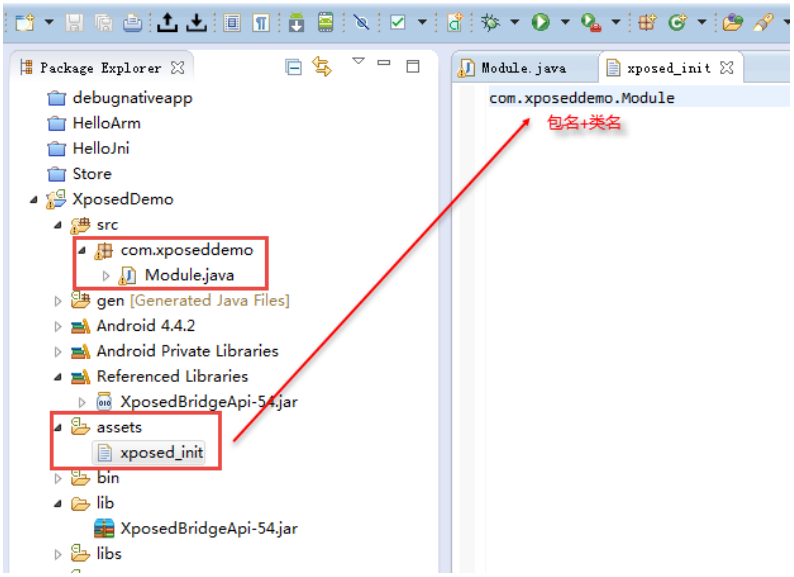
XposedBridgeApi-54.jar 包的下载地址：

http://dl-1.va.us.xda-developers.com/2/7/4/8/8/7/8/XposedBridgeApi-54.jar?key=lqjljy-2L4Sk3lItcQkWoQ&ts=1434529310

3.添加 xposed_init 文件

在 assets 目录下新建一个 xposed_init 文件。

内容为：**包名+类名**，如：com.xposeddemo.Module



4. 在 Hooking Android App 中查找 Hook 的关键点

要被 Hook 的 Android App 就使用非虫大神的书中第 4 章用的 crackme02.apk 应用程序。


16:51
程序未注册
Android程序破解演示实例
用户名：
请输入用户名
注册码：
请输入16位的注册码
注 册


android
annotation
support.v4
com.droider.crackme0201
BuildConfig
MainActivity
R
Manifest
Resources
Certificate
Assembly
Decompiled Java
Strings
Constants
Notes
super.onCreate(savedInstanceState);
this.setContentView(2130903040);
this.setTitle(2131034122);
this.edit_userName = this.findViewById(2131230721);
this.edit_sn = this.findViewById(2131230722);
this.btn_register = this.findViewById(2131230723);
this.btn_register.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
if(!MainActivity.this.checkSN(MainActivity.this.edit_userName.getText().toString().trim(),
MainActivity.this.edit_sn.getText().toString().trim())) {
Toast.makeText(MainActivity.this, 2131034124, 0).show();
}
else {
Toast.makeText(MainActivity.this, 2131034125, 0).show();
MainActivity.this.btn_register.setEnabled(false);
MainActivity.this.setTitle(2131034123);
}
}
});
}
验证注册码的函数checkSN
注册成功

```
private boolean checkSN(String userName, String sn) {
    boolean v7 = false;
    if(userName == null) {
        return v7;
    }

    try {
        if(userName.length() != 0 && sn != null && sn.length() == 16) {
            MessageDigest v1 = MessageDigest.getInstance("MD5");
            v1.reset();
            v1.update(userName.getBytes());
            String v3 = MainActivity.toHexString(v1.digest(), "");
            StringBuilder v5 = new StringBuilder();
            int v4;
            for(v4 = 0; v4 < v3.length(); v4 += 2) {
                v5.append(v3.charAt(v4));
            }

            if(!v5.toString().equalsIgnoreCase(sn)) {
                return v7;
            }

            v7 = true;
        }

        return v7;
    }
    catch(NoSuchAlgorithmException v2) {
        v2.printStackTrace();
        return v7;
    }
}
```

通过对 crackme02.apk 反编译发现只要注册码验证函数 checkSN 返回值是 true 就表明注册码验证通过，我们的 Hook 点就是 checkSN 函数。

5.针对 Hooking Android App 的 Hook 关键点实现 Hook 模块

1). 实现 IXposedHookLoadPackage 接口

2). 确定要 Hook 的 Android App 的包名

3). 判断要 Hook 的包名

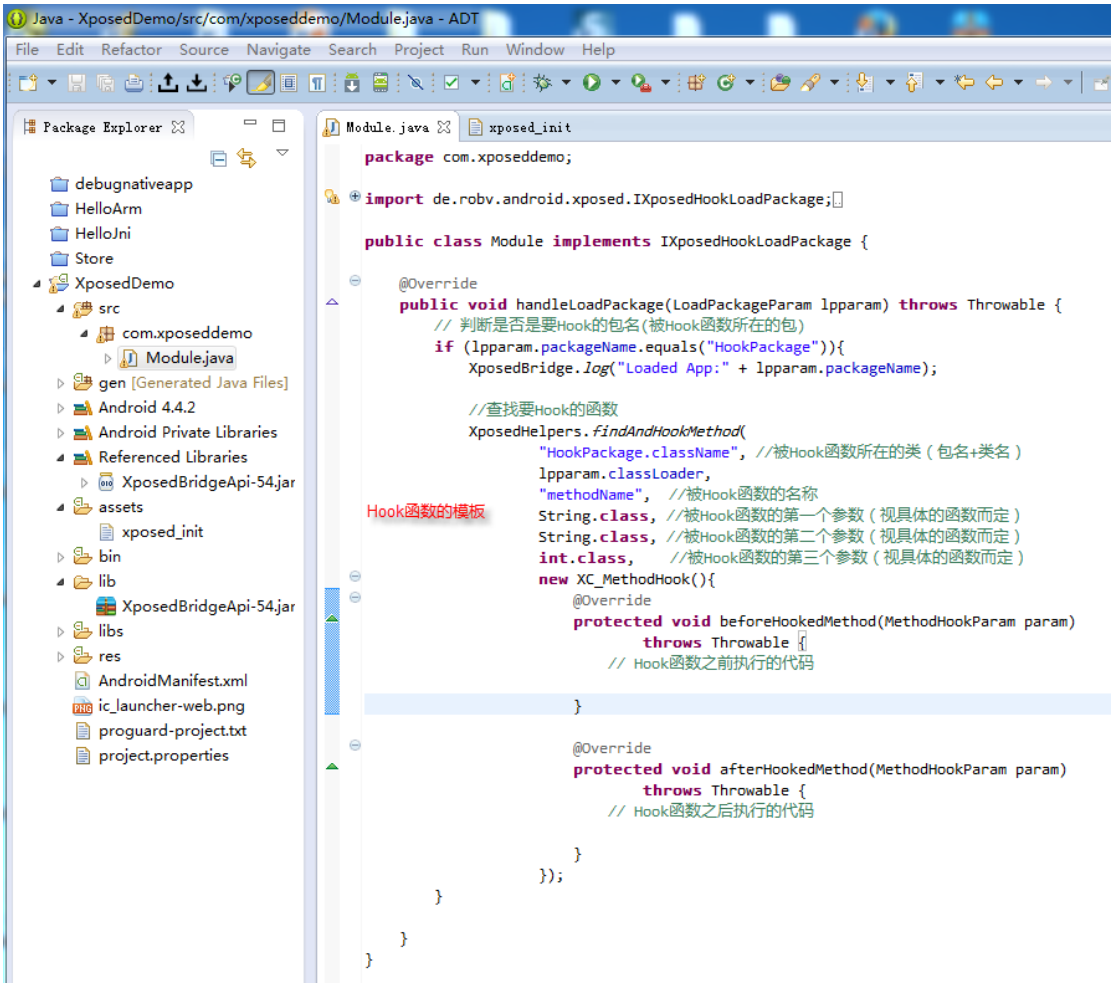
4). 确定要 Hook 的 Android App 的方法

5). 具体实现 Android App 的函数 Hook，调用 XposedHelpers.findAndHookMethod("包名+类名"，lpparam.classLoader, "要 hook 的函数名称"，第一个参数类型，第二个参数类型…... ， new XC_MethodHook() {

```
protectedvoid beforeHookedMethod(MethodHookParam param) {
        //函数执行之前要做的操作
    }

protectedvoid afterHookedMethod(MethodHookParam param) {
        //函数执行之后要做的操作

    }
});
```




```
package com.xposeddemo;

import de.robv.android.xposed.IXposedHookLoadPackage;

public class Module implements IXposedHookLoadPackage {

    @Override
    public void handleLoadPackage(LoadPackageParam lpparam) throws Throwable {
        // 判断是否是要Hook的包名(被Hook函数所在的包)
        if (lpparam.packageName.equals("HookPackage")){
            XposedBridge.log("Loaded App:" + lpparam.packageName);

            //查找要Hook的函数
            XposedHelpers.findAndHookMethod(
                    "HookPackage.className", //被Hook函数所在的类（包名+类名）
                    lpparam.classLoader,
                    "methodName",  //被Hook函数的名称
                    String.class, //被Hook函数的第一个参数（视具体的函数而定）
                    String.class, //被Hook函数的第二个参数（视具体的函数而定）
                    int.class,    //被Hook函数的第三个参数（视具体的函数而定）
                    new XC_MethodHook(){
                        @Override
                        protected void beforeHookedMethod(MethodHookParam param)
                                throws Throwable {
                            // Hook函数之前执行的代码

                        }

                        @Override
                        protected void afterHookedMethod(MethodHookParam param)
                                throws Throwable {
                            // Hook函数之后执行的代码

                        }
                    });
        }

    }
}
```

```
@Override
protected void afterHookedMethod(MethodHookParam param)
        throws Throwable
        {
            // Hook函数之后执行的代码
            param.

        }

    });
```

args : Object[] - XC_MethodHook.MethodHookParam
callbacks : Object[] - XCallback.Param
method : Member - XC_MethodHook.MethodHookParam
thisObject : Object - XC_MethodHook.MethodHookParam
equals(Object o) : boolean - Object
getClass() : Class<?> - Object
getExtra() : Bundle - Param
getObjectExtra(String key) : Object - Param
getResult() : Object - MethodHookParam
getResultOrThrowable() : Object - MethodHookParam

```
place a hook on it
t(Class, String, Ob
```

示例代码的实现：

Manifest | Resources | Certificate | Assembly | Decompiled Java | Strings | Constants | Notes

```
<manifest android:versionCode="1" android:versionName="1.0" package="com.droider.crackme0201" xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />
    <application android:debuggable="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme">
        <activity android:label="@string/title_activity_main" android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

确定了 crackme02.apk 的 Hook 点，我们可以确定要 Hook 函数所在的包为 `com.droider.crackme0201`，要 Hook 函数所在的类为 `com.droider.crackme0201.MainActivity`，要被 Hook 的函数为 `private boolean checkSN(String userName, String sn)`。

见编写的示例代码。

上面这一行代码指定了只有当 com.droider.crackme0201 这个包加载的时候，才会触发一系列的 hook 行为，当这行为触发的时候，de.robv.android.xposed.XposedHelpers 类的 findAndHookMethod 方法就会被调用，并在适当的时候执行前置方法(beforeHookedMethod)和后置方法(afterHookedMethod)，在本次的示例中只要修改 checkSN 函数的返回值即可实现注册的验证绕过。

在 Xposed 框架中安装刚才已经编写好的 Hook 模块 XposedDemo 如上图所示，然后重启手机或者模拟器，我们编写的 XposedDemo 模块在手机或者模拟器重启以后就会生效的，点击运行 crackme02.apk 应程序，按照输入要求输入随意的用户名和注册码该程序就会验证通过（如下图）。

参考的网址：

http://0nly3nd.sinaapp.com/?p=613

http://www.freebuf.com/articles/terminal/56453.html

2015.6.18