

Mechanism.....	2
Zygote.....	2
Hook/Replace.....	2
Developer Wiki.....	2
创建一个 Xposed Module.....	2
寻找目标方法.....	4
使用反射技术定位并 Hook 方法.....	4
进行资源替换.....	5
简易资源替换.....	5
复杂资源.....	6
修改 layouts.....	6
API Reference.....	7
关键类/API 说明.....	7
IXposedHookLoadPackage.....	7
XposedHelpers.....	7
辅助项 API.....	8
XposedBridge 类.....	8
XposedHelpers 类.....	9

Mechanism

Zygote

Dalvik 孵化器 Zygote 进程对应的程序是/system/bin/app_process，在 Android Framework 载入的时候，一个我们自己定义的 app_process 程序会被复制到/system/bin 目录下。这个程序会在开机的时候添加了额外的 jar 文件到 classpath 从而实现可以在 Zygote 上下文中调用这些方法。

这个 jar 文件被放置在了/data/xposed/XposedBridge.jar 目录下，Zygote 进程会首先调用这个 jar 文件中的方法。XposedBridge 有一个私有的 Native (JNI) 方法 hookMethodNative，这个方法也在 app_process 中使用。这个函数提供一个方法对象利用 Java 的 Reflection 机制来对内置方法覆写。

Configuration	RequireMent
Root Access	因为 Xposed 工作原理是在/system/bin 目录下替换文件，在 install 的时候需要 root 权限，但是运行时不需要 root 权限。
版本要求	需要在 Android 4.0 以上版本的机器中

Hook/Replace

Xposed 框架中真正起作用的是对方法的 hook。在 Repackage 技术中，如果要对 APK 做修改，则需要修改 Smali 代码中的指令。而另一种动态修改指令的技术需要在程序运行时基于匹配搜索来替换 smali 代码，但因为方法声明的多样性与复杂性，这种方法也比较复杂。

XposedBridge 这个 jar 包含有一个私有的本地方法：hookMethodNative，该方法在附加的 app_process 程序中也得到了实现。它将一个方法对象作为输入参数（你可以使用 Java 的反射机制来获取这个方法）并且改变 Dalvik 虚拟机中对于该方法的定义。它将该方法的类型改变为 native 并且将这个方法的实现链接到它的本地的通用类的方法。换言之，当调用那个被 hook 的方法时候，通用的类方法会被调用而不会对调用者有任何的影响。在 hookMethodNative 的实现中，会调用 XposedBridge 中的 handleHookedMethod 这个方法来传递参数。

当多模块同时 Hook 一个方法的时候，Xposed 会自动根据 Module 的优先级来排序，调用顺序如下：

A.before -> B.before -> original method -> B.after -> A.after

Developer Wiki

创建一个 Xposed Module

一个 XposedModule 本质上是设定了部分特殊元数据标志位的普通应用程序，需要在

AndroidManifest.xml 文件中添加如下设置：

AndroidManifest.xml => Application => Application Nodes (at the bottom) => Add => Meta Data

添加节点：name = xposedmodule，value = true。name = xposedminversion，value = API level。

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="de.robv.android.xposed.mods.tutorial"

    android:versionCode="1"

    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

    <application

        android:icon="@drawable/ic_launcher"

        android:label="@string/app_name" >

        <meta-data android:value="true" android:name="xposedmodule"/>

        <meta-data android:value="2.0*" android:name="xposedminversion"/>

        <meta-data android:value="Demonstration of the Xposed framework.\nMakes the status bar clock red."
        android:name="xposeddescription"/>

    </application>

</manifest>
```

然后，将 XposedBridge.jar 这个引用导入到工程中，加入到 reference path 中。

下面开始创建一个新的工程：

```
package com.kevin.myxposed;

import android.util.Log;

import de.robv.android.xposed.IXposedHookLoadPackage;

import de.robv.android.xposed.XposedBridge;

import de.robv.android.xposed.callbacks.XC_LoadPackage.LoadPackageParam;

public class XposedInterface implements IXposedHookLoadPackage {

    public void handleLoadPackage(final LoadPackageParam lpparam) throws Throwable {

        XposedBridge.log("Kevin-Loaded app: " + lpparam.packageName);

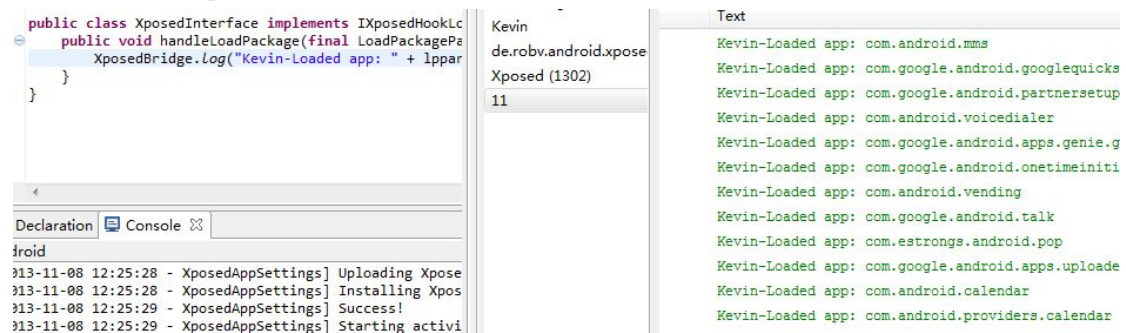
    }

}
```

然后在 assets 目录下新建一个 xposed_init 文件，这个文件声明了需要加载到 XposedInstaller 的入口类：

com.kevin.myxposed.XposedInterface

运行程序并在 XposedInstaller 的 Module 选项中激活，重启机器后可以得到如下数据：



The screenshot shows the XposedInstaller interface. On the left, the 'XposedInterface' module is selected, showing its code. The code defines a public class XposedInterface implementing IXposedHookLoadPackage, with a handleLoadPackage method that logs 'Kevin-Loaded app: ' followed by the package name. The bottom panel shows the 'Declaration' tab with the module's declaration and a console log showing the installation process. On the right, a list of loaded applications is displayed, including com.android.mms, com.google.android.googlequicksearch, com.google.android.partnersetup, com.android.voicedialer, com.google.android.apps.genie.g, com.google.android.onetimeinit, com.android.vending, com.google.android.talk, com.ezstrongs.android.pop, com.google.android.apps.upload, com.android.calendar, and com.android.providers.calendar.

寻找目标方法

- 反编译目标程序，查看 Smali 代码。
- 直接在 AOSP 中查看。

使用反射技术定位并 Hook 方法

在上一步中我们已经定位了需要 Hook 的方法以及所在的类，譬如：

com.android.systemui.statusbar.policy.Clock 类

中的 updateClock 方法。

```
package de.robv.android.xposed.mods.tutorial;

import static de.robv.android.xposed.XposedHelpers.findAndHookMethod;
import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.callbacks.XC_LoadPackage.LoadPackageParam;

public class Tutorial implements IXposedHookLoadPackage {

    public void handleLoadPackage(final LoadPackageParam lpparam) throws Throwable {
        if (!lpparam.packageName.equals("com.android.systemui"))
            return;

        findAndHookMethod("com.android.systemui.statusbar.policy.Clock", lpparam.classLoader, "handleUpdateClock", new
XC_MethodHook() {
            @Override
            protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                // this will be called before the clock was updated by the original method
            }
            @Override
            protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                // this will be called after the clock was updated by the original method
            }
        });
    }
}
```

```
    }  
    });  
    }  
}
```

关于 `findAndHookMethod` 方法的说明见下面的 *API Reference*。

进行资源替换

简易资源替换

下面所使用的方法可以适用于 `Boolean`、`Color`、`Integer`、`int[]`、`String` 与 `String[]`。

其中，对于 Android 框架层的资源（所有的 APP 都需要调用的资源）应该在 `initZygote` 这个方法中完成替换。而对于属于应用程序的资源，应该在 `hookInitPackageResources` 这个方法中完成替换。

```
@Override  
public void initZygote(IXposedHookZygoteInit.StartupParam startupParam) throws Throwable {  
    XResources.setSystemWideReplacement("android", "bool", "config_unplugTurnsOnScreen", false);  
}  
  
@Override  
public void handleInitPackageResources(InitPackageResourcesParam resparam) throws Throwable {  
    // replacements only for SystemUI  
    if (!resparam.packageName.equals("com.android.systemui"))  
        return;  
  
    // different ways to specify the resources to be replaced  
    resparam.res.setReplacement(0x7f080083, "YEAH!"); // WLAN toggle text. You should not do this because the id is not  
    // fixed. Only for framework resources, you could use android.R.string.something  
  
    resparam.res.setReplacement("com.android.systemui:string/quickpanel_bluetooth_text", "WOO!");  
  
    resparam.res.setReplacement("com.android.systemui", "string", "quickpanel_gps_text", "HOO!");  
  
    resparam.res.setReplacement("com.android.systemui", "integer", "config_maxLevelOfSignalStrengthIndicator", 6);  
  
    resparam.res.setReplacement("com.android.systemui",  
        "drawable", "status_bar_background",  
        new XResources.DrawableLoader() {  
            @Override  
            public Drawable newDrawable(XResources res, int id) throws Throwable {  
                return new ColorDrawable(Color.WHITE);  
            }  
        });  
}
```

```
}
```

复杂资源

```
package de.robv.android.xposed.mods.coloredcirclebattery;

import android.content.res.XModuleResources;
import de.robv.android.xposed.IXposedHookInitPackageResources;
import de.robv.android.xposed.IXposedHookZygoteInit;
import de.robv.android.xposed.callbacks.XC_InitPackageResources.InitPackageResourcesParam;

public class ColoredCircleBattery implements IXposedHookZygoteInit, IXposedHookInitPackageResources {
    private static String MODULE_PATH = null;

    @Override
    public void initZygote(StartupParam startupParam) throws Throwable {
        MODULE_PATH = startupParam.modulePath;
    }

    @Override
    public void handleInitPackageResources(InitPackageResourcesParam resparam) throws Throwable {
        if (!resparam.packageName.equals("com.android.systemui"))
            return;

        XModuleResources modRes = XModuleResources.createInstance(MODULE_PATH, resparam.res);
        resparam.res.setReplacement("com.android.systemui", "drawable", "stat_sys_battery",
modRes.fwd(R.drawable.battery_icon));
        resparam.res.setReplacement("com.android.systemui", "drawable", "stat_sys_battery_charge",
modRes.fwd(R.drawable.battery_icon_charge));
    }
}
```

修改 layouts

```
@Override
public void handleInitPackageResources(InitPackageResourcesParam resparam) throws Throwable {
    if (!resparam.packageName.equals("com.android.systemui"))
        return;

    resparam.res.hookLayout("com.android.systemui", "layout", "status_bar", new XC_LayoutInflated() {
        @Override
        public void handleLayoutInflated(LayoutInflatedParam liparam) throws Throwable {
            TextView clock = (TextView) liparam.view.findViewById(
```

君子务本 天道酬勤

NUPT&Cimer 王下邀月熊(kevin)

```
        lpparam.res.getIdentifier("clock", "id", "com.android.systemui"));
        clock.setTextColor(Color.RED);
    }
});
}
```

API Reference

关键类/API 说明

IXposedHookLoadPackage

Method	Description
<i>handleLoadPackage</i>	<p>这个方法用于在加载应用程序的包的时候执行用户的操作。</p> <ul style="list-style-type: none">调用示例 <pre>public class XposedInterface implements IXposedHookLoadPackage { public void handleLoadPackage(final LoadPackageParam lpparam) throws Throwable { XposedBridge.log("Kevin-Loaded app: " + lpparam.packageName); } }</pre> <ul style="list-style-type: none">参数说明 <p><i>final LoadPackageParam lpparam</i> 这个参数包含了加载的应用程序的一些基本信息。</p>

XposedHelpers

Method	Description
--------	-------------

<code>findAndHookMethod</code>	<p>这是一个辅助方法，可以通过如下方式静态导入：</p> <pre>import static de.robv.android.xposed.XposedHelpers.findAndHookMethod;</pre> <ul style="list-style-type: none">● 使用示例 <pre>findAndHookMethod("com.android.systemui.statusbar.policy.Clock", lpparam.classLoader, "handleUpdateClock", new XC_MethodHook() { @Override protected void beforeHookedMethod(MethodHookParam param) throws Throwable { // this will be called before the clock was updated by the original method } @Override protected void afterHookedMethod(MethodHookParam param) throws Throwable { // this will be called after the clock was updated by the original method } });</pre> <ul style="list-style-type: none">● 参数说明<ul style="list-style-type: none">❖ <code>findAndHookMethod(Class<?> clazz, //需要 Hook 的类名</code> <code>ClassLoader, //类加载器，可以设置为 null</code> <code>String methodName, //需要 Hook 的方法名</code> <code>Object... parameterTypesAndCallback</code> <p>该函数的最后一个参数集，包含了：</p> <p>(1) Hook 的目标方法的参数,譬如：</p> <pre>"com.android.internal.policy.impl.PhoneWindow.DecorView"</pre> <p>是方法的参数的类。</p> <p>(2) 回调方法：</p> <ul style="list-style-type: none">a.XC_MethodHookb.XC_MethodReplacement
	<ul style="list-style-type: none">●

辅助项 API

Xposed 框架也为我们提供了很多的辅助项来帮助我们快速开发 XposedModule。

XposedBridge 类

Method	Description
<code>log</code>	该方法可以将 log 信息以及 Throwable 抛出的异常信息输出到标准的 logcat 以及/data/xposed/debug.log 这个文件中。
<code>hookAllMethods /</code>	该方法可以用来 hook 某个类中的所有方法或者构造函数，但是不同的

hookAllConstructors	Rom（非 Android 原生 Rom）会有不同的变种。
---------------------	-------------------------------

XposedHelpers 类

这个类用的也是比较多，可以使用

Window => Preferences => Java => Editor => Content Assist => Favorites => New Type, enter de.robv.android.xposed.XposedHelpers

这种方式将 XposedHelpers 这个类加入到 Eclipse 静态调用中方便查阅。

Method	Description
findMethod / findConstructor / findField	这是一组用于检索方法的方法。
callMethod / callStaticMethod / newInstance	
assetAsByteArray	以字节数组的形式返回 asset，可以以如下方式调用： <pre>public class XposedTweakbox { private static final String MODULE_PATH = null; // injected by XposedBridge public static void init(String startClassName) throws Exception { if (startClassName != null) return; Resources tweakboxRes = XModuleResources.createInstance(MODULE_PATH, null); byte[] crtPatch = assetAsByteArray(tweakboxRes, "crtfix_samsung_d506192d5049a4042fb84c0265edfe42.bsdiff"); ... } }</pre>
getMD5Sum	返回对于一个文件的 MD5 校验值，需要 root 权限。
getProcessPid	获取一个进程的 PID 值，输入参数为 <i>/proc/[pid]/cmdline</i>