

# 离我的App远点儿! - Android应用劫持的攻与防

SESSION ID: MAN-W07

周荣誉

资深安全研究员  
百度  
微博: @colordancer



# 议程

- ◆ Root or Not Root
- ◆ App劫持
- ◆ Hook详解
- ◆ App劫持演示
- ◆ 注入和Hook检测/修复
- ◆ 离我的App远点儿！——创建可信App运行环境

**RSAC** CONFERENCE 2014  
ASIA PACIFIC & JAPAN



**Root or Not Root**

# Root or Not Root

- ◆ 这不是我们今天要讨论的问题
- ◆ 不过这个问题其实很简单
  - ◆ 需要：卸载预装、控制自启动、个性化、.....
  - ◆ 不需要：不安全！
- ◆ 我们需要的是“安全的Root”



# Root的危害

- ◆ Android通过用户隔离来保障每个应用的安全
- ◆ 每个App都可以申请Root权限
- ◆ Root后，你的App就可以被其他App访问
  - ◆ 内存被篡改
  - ◆ 行为被监控
  - ◆ .....

```
u0_a5 936 181 889740 46828 ffffffff 00000000 S android.process.media
u0_a48 963 181 884940 47140 ffffffff 00000000 R com.google.android.inputmethod.pinyin
u0_a7 987 181 1018824 53324 ffffffff 00000000 S com.google.android.gms
radio 1002 181 886732 39532 ffffffff 00000000 S com.android.phone
radio 1014 181 868940 32092 ffffffff 00000000 S com.redbend.vdmc
nfc 1027 181 884840 36536 ffffffff 00000000 S com.android.nfc
u0_a20 1041 181 1045096 93236 ffffffff 00000000 S com.google.android.googlequicksearchbox
u0_a7 1153 181 902264 50052 ffffffff 00000000 S com.google.process.location
u0_a7 1175 181 911244 53672 ffffffff 00000000 S com.google.process.gapps
root 1259 1 7212 492 ffffffff 00000000 S /system/bin/mpdecision
dhcp 1391 1 1020 476 ffffffff 00000000 S /system/bin/dhccpd
u0_a101 1412 181 897728 56624 ffffffff 00000000 S com.lbe.security:service
u0_a101 1474 1412 1052 244 ffffffff 00000000 S lbsec.monitor
```

```

@hammerhead:/ $ su
hammerhead:/ # ps | grep chrome
02  1584  180  1156420 106048 ffffffff 4006773c S com.android.chrome
0  2370  180  1223960 71560 ffffffff 4006773c S com.android.chrome:sandboxed_process0
hammerhead:/ # cat /proc/1584/maps | grep /data/
00000000-00000000 r--s 00017000 b3:1c 82184 /data/data/de.robv.android.xposed.installer/bin/XposedBridge.jar
00000000-00000000 r--p 00000000 b3:1c 106000 /data/dalvik-cache/data@data@de.robv.android.xposed.installer@bin@
00000000-00000000 r--p 0000e000 b3:1c 106000 /data/dalvik-cache/data@data@de.robv.android.xposed.installer@bin@
00000000-00000000 r--p 0000f000 b3:1c 106000 /data/dalvik-cache/data@data@de.robv.android.xposed.installer@bin@
00000000-00000000 r--p 0001f000 b3:1c 106000 /data/dalvik-cache/data@data@de.robv.android.xposed.installer@bin@
00000000-00000000 r--p 00020000 b3:1c 106000 /data/dalvik-cache/data@data@de.robv.android.xposed.installer@bin@
00000000-00000000 r--s 01c7e000 b3:1c 162902 /data/app/com.android.chrome-1.apk
00000000-00000000 r--p 00000000 b3:1c 106021 /data/dalvik-cache/data@app.com.android.chrome-1.apk@classes.dex
00000000-00000000 r--s 01c7e000 b3:1c 162902 /data/app/com.android.chrome-1.apk
00000000-00000000 r--s 01a71000 b3:1c 162902 /data/app/com.android.chrome-1.apk
00000000-00000000 r-xp 00000000 b3:1c 40723 /data/app-lib/com.android.chrome-1/libchrome.1847.114.so
00000000-00000000 r--p 02238000 b3:1c 40723 /data/app-lib/com.android.chrome-1/libchrome.1847.114.so
00000000-00000000 rwxp 02388000 b3:1c 40723 /data/app-lib/com.android.chrome-1/libchrome.1847.114.so
00000000-00000000 rw-p 02389000 b3:1c 40723 /data/app-lib/com.android.chrome-1/libchrome.1847.114.so
00000000-00000000 rwxp 0239d000 b3:1c 40723 /data/app-lib/com.android.chrome-1/libchrome.1847.114.so
00000000-00000000 rw-p 0239e000 b3:1c 40723 /data/app-lib/com.android.chrome-1/libchrome.1847.114.so
00000000-00000000 r--s 00000000 b3:1c 81805 /data/data/com.android.chrome/app_chrome/paks/zh-CN.pak
00000000-00000000 r--s 00000000 b3:1c 81801 /data/data/com.android.chrome/app_chrome/paks/chrome_100_percent.pak
00000000-00000000 r--s 00000000 b3:1c 81804 /data/data/com.android.chrome/app_chrome/paks/resources.pak
00000000-00000000 r--s 00000000 b3:1c 82120 /data/data/com.android.chrome/files/tab6
00000000-00000000 r--s 00000000 b3:1c 82700 /data/data/com.android.chrome/files/tab20
00000000-00000000 r--s 00019000 b3:1c 97945 /data/data/com.lbe.security/app_hips/client.jar
00000000-00000000 r-xp 00000000 b3:1c 98005 /data/data/com.lbe.security/app_hips/liblbeclient.so
00000000-00000000 r--p 0000e000 b3:1c 98005 /data/data/com.lbe.security/app_hips/liblbeclient.so
00000000-00000000 rw-p 0000f000 b3:1c 98005 /data/data/com.lbe.security/app_hips/liblbeclient.so
00000000-00000000 r--p 00000000 b3:1c 106036 /data/dalvik-cache/data@data@com.lbe.security@app_hips@client.jar@
00000000-00000000 r--s 00000000 b3:1c 82101 /data/data/com.android.chrome/files/tab0
00000000-00000000 r--s 00000000 b3:1c 82116 /data/data/com.android.chrome/files/tab1

```

## 被注入后的Chrome进程

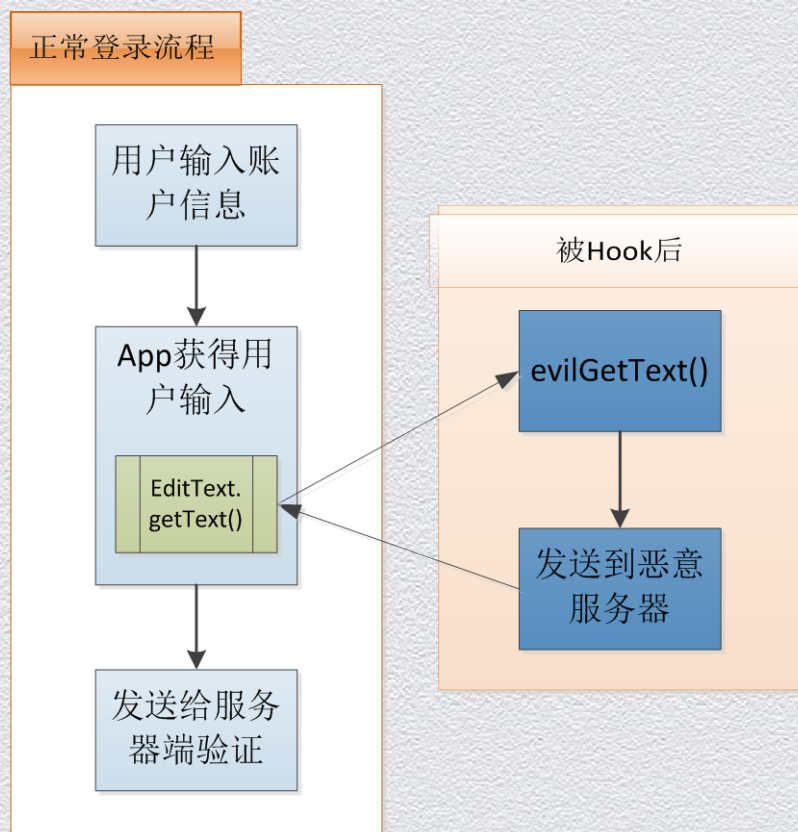
**RSAC** CONFERENCE **2014**  
ASIA PACIFIC & JAPAN



App劫持

# 什么是App劫持?

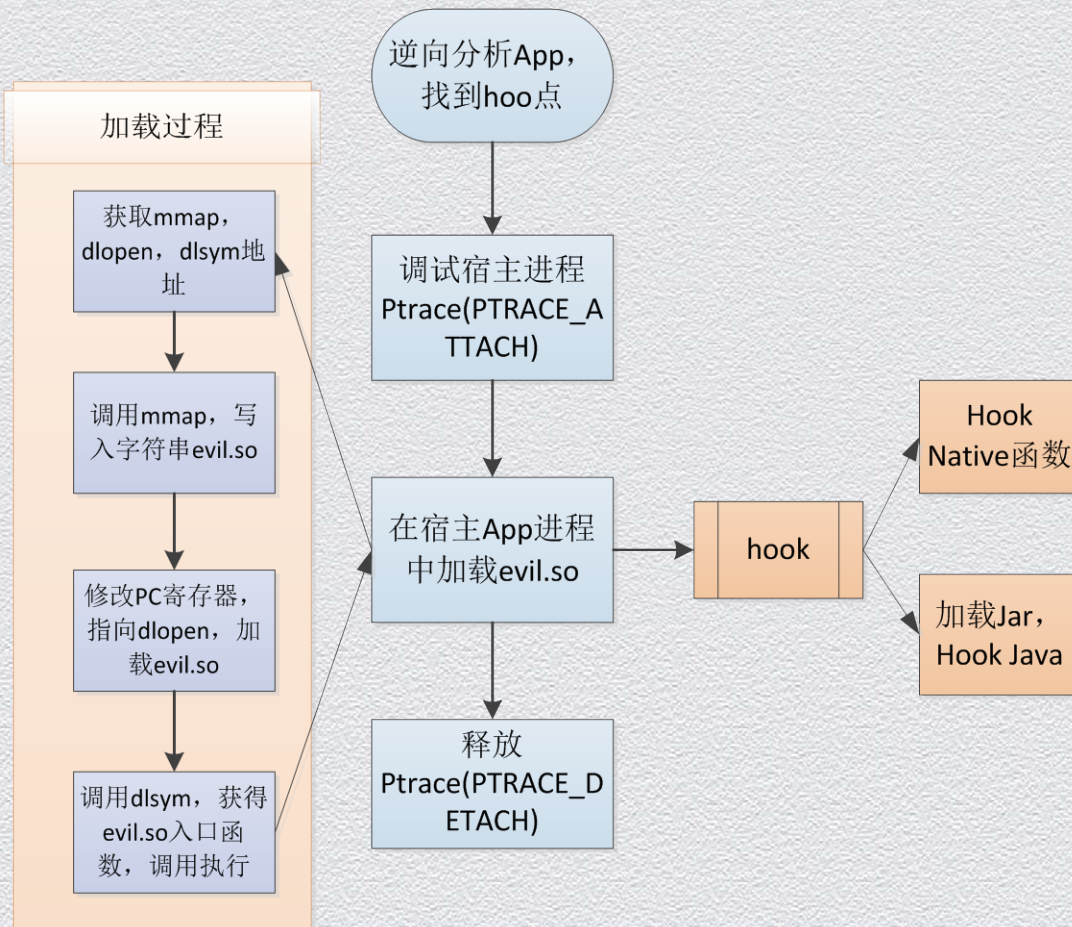
- ◆ App劫持: App的执行流程被重定向
- ◆ 通常通过“注入+Hook”来实现





# App劫持过程

- ◆ Step1: 逆向分析App的逻辑
- ◆ Step2: 注入模块到App进程中
  - ◆ *Ptrace*
  - ◆ *Dlopen*
- ◆ Step3: Hook
  - ◆ Java Hook
  - ◆ Native(so) Hook



## 注入Hook过程

**RSAC** CONFERENCE **2014**  
ASIA PACIFIC & JAPAN



## Hook详解

# 具有Hook行为的App

- ◆ 病毒
  - ◆ 数量较少
    - ◆ 开发hook模块成本较高
    - ◆ 需要root
- ◆ 安全软件
  - ◆ 主动防御

# Hook类病毒

## ◆ Wind Seeker



```
v12 = JNIEnv;  
v13 = (*(int (__fastcall **)(_DWORD *, int, _DWORD, _DWORD))(*JNIEnv + 452))(  
    JNIEnv,  
    v11,  
    "getSystemClassLoader",  
    "()Ljava/lang/ClassLoader;");  
v14 = _JNIEnv::CallStaticObjectMethod(v12, v11, v13);  
v15 = _JNIEnv::NewObject(JNIEnv);  
_android_log_print(6, "FOR_AD", "DexClassLoader= %p", v15, v10, 0, v14);  
v16 = (*(int (**)(void))(*JNIEnv + 132))();  
_android_log_print(6, "FOR_AD", "loadClass= %p", v16);  
(* (void (**)(void))(*JNIEnv + 668))();  
v17 = _JNIEnv::CallObjectMethod(JNIEnv, v15, v16);  
_android_log_print(6, "FOR_AD", "Class= %p", v17);  
v18 = (*(int (**)(void))(*JNIEnv + 452))();  
_android_log_print(6, "FOR_AD", "setAppHook = %p", v18);  
_JNIEnv::CallStaticVoidMethod(JNIEnv, v17, v18);
```

```
long l3 = localCursor.getLong(localCursor.getColumnIndex("_id"));  
String str8 = localCursor.getString(localCursor.getColumnIndex("selfuin"));  
String str9 = localCursor.getString(localCursor.getColumnIndex("frienduin"));  
String str10 = localCursor.getString(localCursor.getColumnIndex("senderuin"));  
long l4 = localCursor.getLong(localCursor.getColumnIndex("time"));  
String str11 = localCursor.getString(localCursor.getColumnIndex("msg"));  
localCursor.getInt(localCursor.getColumnIndex("msgtype"));  
localCursor.getInt(localCursor.getColumnIndex("isread"));  
int n = localCursor.getInt(localCursor.getColumnIndex("issend"));  
localCursor.getLong(localCursor.getColumnIndex("msgseq"));  
localCursor.getLong(localCursor.getColumnIndex("shmsgseq"));  
localCursor.getInt(localCursor.getColumnIndex("istroop"));  
localCursor.getInt(localCursor.getColumnIndex("extraflag"));  
String str12 = localCursor.getString(localCursor.getColumnIndex("friendnick"));
```

# 安全类App

主流安全App:

其中含有注入行为的:



# Hook类型

- ◆ Java Hook
  - ◆ Static Field Hook: 静态成员hook
  - ◆ Method Hook: 函数hook
- ◆ Native So Hook
  - ◆ GOT Hook: 全局偏移表hook
  - ◆ SYM Hook: 符号表hook
  - ◆ Inline Hook: 函数内联hook

# Java静态成员Hook

- ◆ 修改Java Class的静态成员的值
- ◆ 通过反射
  - ◆ `Class<?> cls = Class.forName("")`
  - ◆ `Field fld = cls.getDeclaredField("")`
- ◆ 为什么是静态成员static field?
  - ◆ 反射只能得到Class，不能得到object



```

private static void a(Object arg3) {
    if(arg3 != null) {
        Object v0 = b.reflectField(KSCONST.decrypt("android.webkit.BrowserFrame$ConfigCallback")
            , KSCONST.decrypt("mHandlers"), arg3);
        if(v0 != null) {
            b.setField(KSCONST.decrypt("android.webkit.BrowserFrame$ConfigCallback"), KSCONST.decrypt(
                ("mHandlers"), arg3, new at(((ArrayList)v0)));
        }
    }
}

public static Object setField(String arg4, String arg5, Object arg6, Object arg7) {
    Exception v1_1;
    Object v0_1;
    Field v2;
    Object v1 = null;
    try {
        v2 = Class.forName(arg4).getDeclaredField(arg5);
        v2.setAccessible(true);
        v0_1 = v2.get(arg6);
    }
    catch(Exception v0) {
        Exception v3 = v0;
        v0_1 = v1;
        v1_1 = v3;
        goto label_12;
    }

    try {
        v2.set|(arg6, arg7);
        goto label_7;
    }
    catch(Exception v1_1) {
    }

label_12:
    v1_1.printStackTrace();
label_7:
    return v0_1;
}

```

## Java static field hook示例

# Java函数hook

- ◆ 修改Java类的某函数指向自定义函数
  - ◆ Dalvik: Java Method -> Native Method
  - ◆ ART: Method Inline Hook

# Java函数hook

- ◆ Dalvik:
  - ◆ Java函数在内存中指向的是字节码: *Method->insns*
  - ◆ 字节码inline hook? 复杂!
  - ◆ 将Java函数转为Native函数
    - ◆ *Method->nativeFunc* 指向So中的函数

```
// Replace method with our own code
SET_METHOD_FLAG(method, ACC_NATIVE);
method->nativeFunc = &xposedCallHandler;
method->insns = (const u2*) hookInfo;
method->registersSize = method->insSize;
method->outsSize = 0;
```

# So GOT Hook

- ◆ 修改App加载的so中Got表里函数的绝对地址
- ◆ GOT
  - ◆ Global Offset Table: 全局偏移表
  - ◆ 存储了依赖库so的函数的绝对地址
- ◆ PLT
  - ◆ Procedure Linkage Table: 过程链接表
  - ◆ 将当前so内对依赖库函数的相对调用转移到Got中的绝对地址

# So GOT Hook

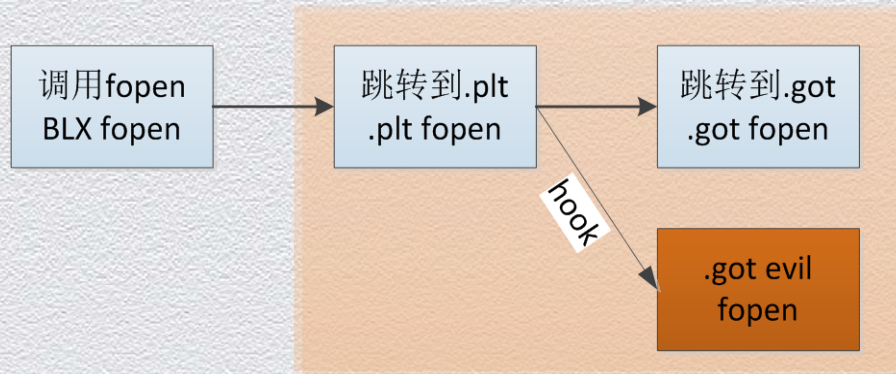
## ◆ 正常执行流程

```
.text:00008C92 02 98          LDR    R0, [SP,#0x640+filename] ; filename
.text:00008C94 79 44          ADD    R1, PC          ; "W+"
.text:00008C96 FF F7 E8 ED    BLX   fopen

; FILE *fopen(const char *filename, const char *modes)
.plt:00008868          ; fopen
.plt:00008868          ; CODE XREF: sub_8B90+12↓p
.plt:00008868          ; sub_8BE8+AE↓p
.plt:00008868 00 C6 8F E2    ADR    R12, 0x8870
.plt:0000886C 03 CA 8C E2    ADD    R12, R12, #0x3000
.plt:00008870 2C F7 BC E5    LDR    PC, [R12,#(fopen_ptr - 0xB870)]! ; __imp_fopen

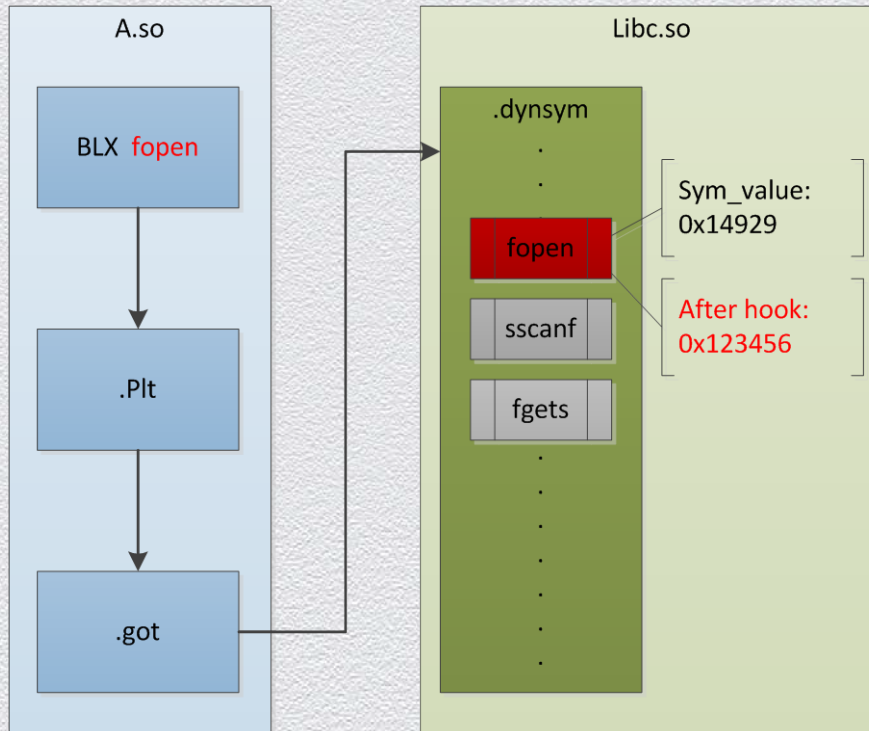
extern:0000C120          ; FILE *fopen(const char *filename, const char *modes)
extern:0000C120 00 00 00 00    IMPORT __imp_fopen      ; CODE XREF: fopen+8↑j
```

## ◆ Hook后执行流程



# So SYM Hook

- ◆ 修改App加载的So的符号表里的函数地址
- ◆ 缺点：
  - ◆ 必须在so加载前hook

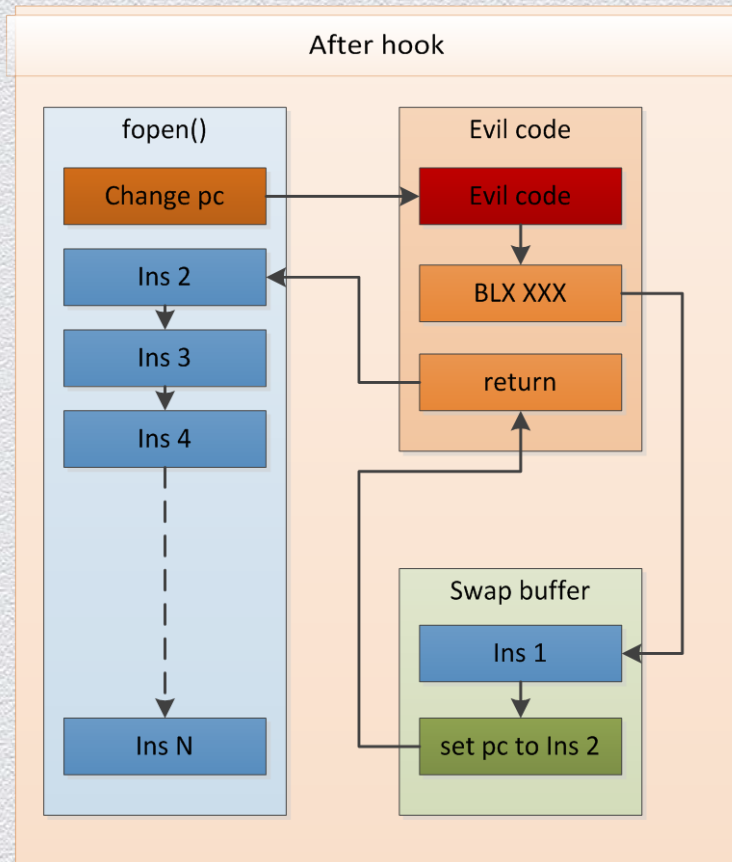
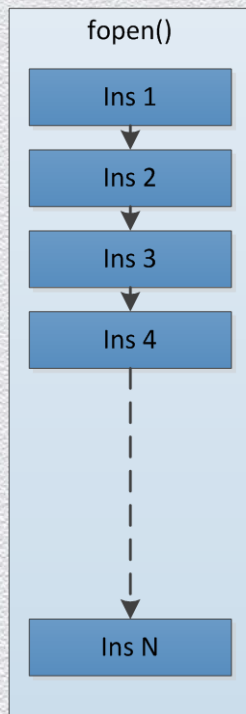


## So SYM hook流程

# So Inline Hook

- ◆ 修改App加载的So的函数里的代码
- ◆ 跳转有范围限制，需要跳板
- ◆ 需要兼容arm和thumb指令
- ◆ 优点：覆盖面广
- ◆ 缺点：难度高，不稳定





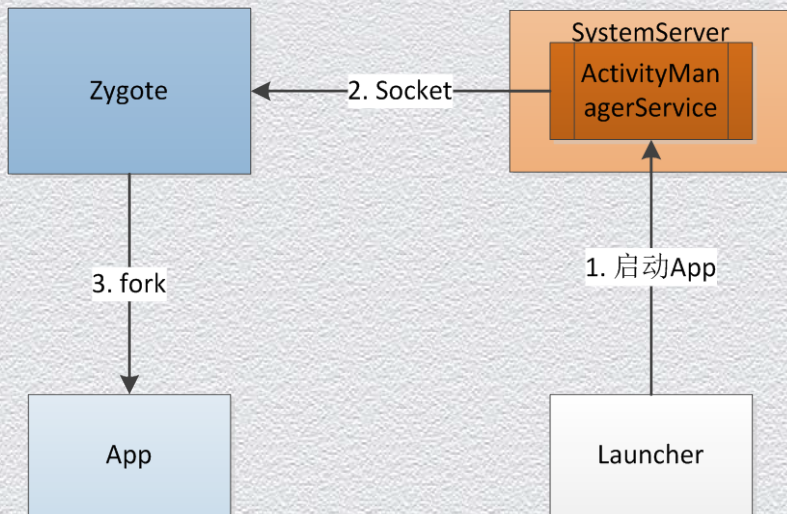
So inline Hook流程

# Hook Zygote

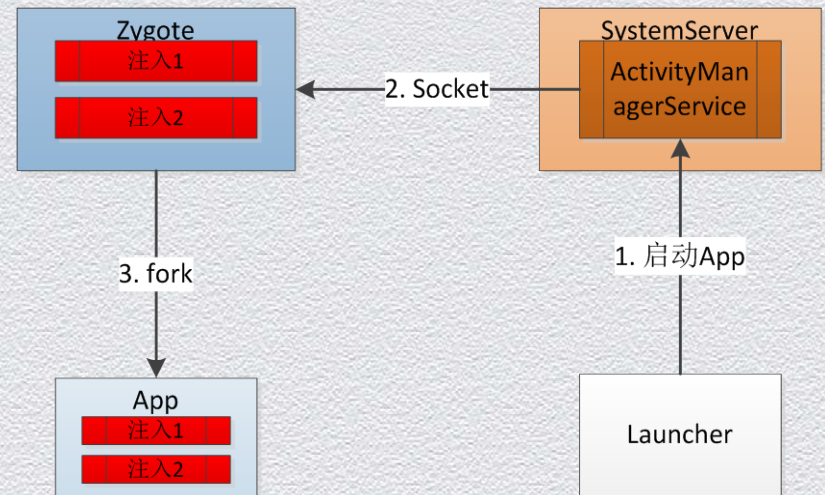
- ◆ 全局hook的方法：
  - ◆ 😞 Hook当前已运行的所有App进程，监听新进程
  - ◆ 😊 Hook Zygote进程
- ◆ App进程是有Zygote孵化出来的
  - ◆ 如果Zygote被注入/hook，其他App都将被注入/hook

# Hook Zygote

## Hook前



## Hook后



# 流行的Hook框架

- ◆ Xposed
  - ◆ <http://repo.xposed.info/>
- ◆ Cydia Substrate
  - ◆ <http://www.cydiasubstrate.com/>
- ◆ ADBI/DDI
  - ◆ <https://github.com/crmulliner/adbi>
  - ◆ <https://github.com/crmulliner/ddi>

# 流行的Hook框架：Xposed

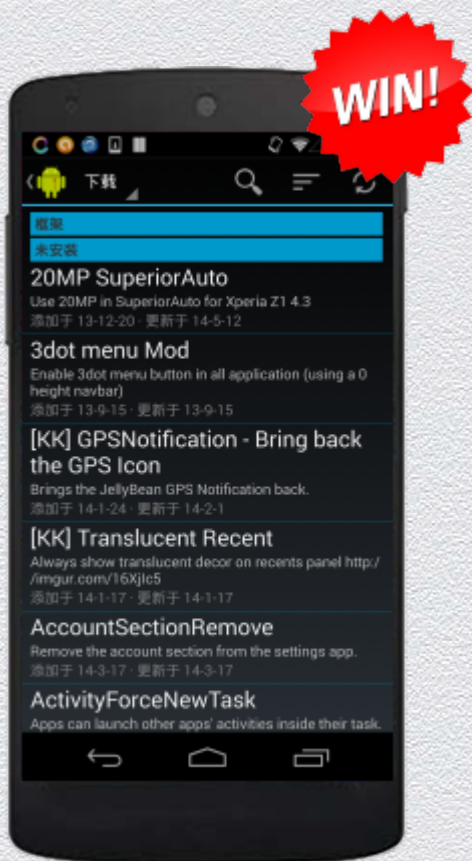
- ◆ 目前使用最广的Android平台开源hook框架
- ◆ 替换App\_Process
- ◆ Java函数转为Native函数
- ◆ 在被hook函数执行时，调用BeforeHook/AfterHook接口
- ◆ 添加/删除hook需要重启zygote

# 流行的Hook框架：Cydia Substrate

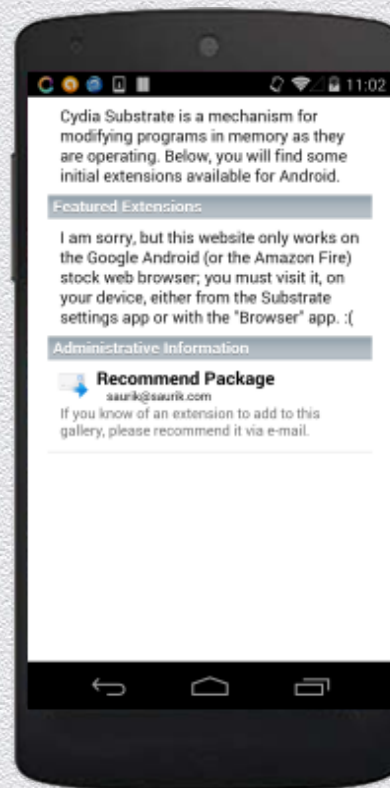
- ◆ 兼容iOS和Android
- ◆ 在iOS平台使用广泛
- ◆ 需要重启Zygote

# 流行的Hook框架对比

## Xposed



## Cydia Substrate



**RSAC** CONFERENCE 2014  
ASIA PACIFIC & JAPAN



App劫持演示





## Demo: Facebook登陆劫持



## Demo: 招行网银WebView钓鱼

**RSAC** CONFERENCE 2014  
ASIA PACIFIC & JAPAN

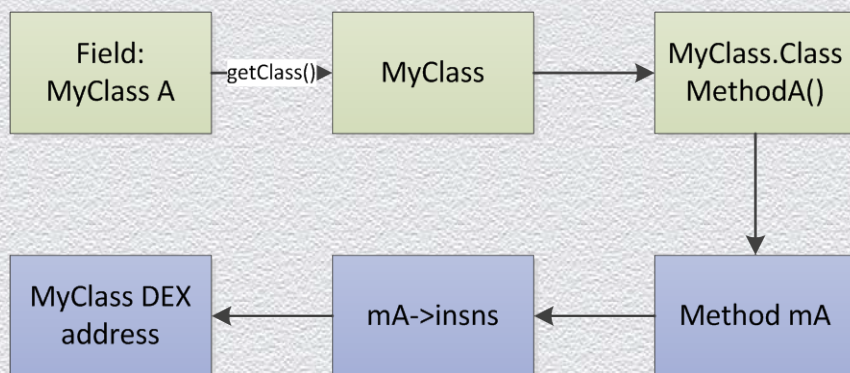


App劫持检测/修复

# Hook检测/修复: Java Hook

## ◆ 静态成员hook

- ◆ 检测: 根据变量的class判定是否属于当前App的classes.dex
- ◆ 修复: 无法修复: 无法获得原始值



# Hook检测/修复: Java Hook

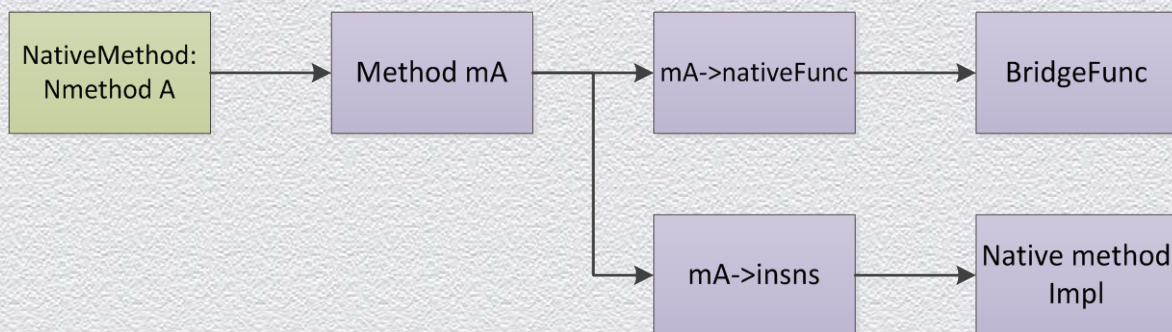
## ◆ 函数hook

### ◆ Dalvik: Java->Native

#### ◆ 检测:

- ◆ 根据 *nativFunc* 找到对应的 *BridgeFunction*, 判定其是否在 *libdvm.so* 内存区间;
- ◆ 判断 *insns* 指向的 *so* 内存区间

#### ◆ 修复: 无法修复, *Native->Java* 不可逆



# Hook检测/修复: So Hook

## ◆ GOT hook

- ◆ 检测: 判断GOT地址指向的SO是否属于当前App
- ◆ 修复: 获取函数的正确地址, 然后写回

## ◆ SYM hook

- ◆ 检测: 判断Symbol地址指向的SO是否属于当前App
- ◆ 修复: 获取Symbol的正确地址, 然后写回

## ◆ Inline hook

- ◆ 检测: 指令解析
- ◆ 修复: 从磁盘上读取函数指令, 然后写回

# 注入修复

- ◆ 为什么要修复注入？
  - ◆ 有可能只注入、不hook：监听线程
  - ◆ Hook：摘除钩子
- ◆ 修复注入的方法
  - ◆ munmap
  - ◆ 必须先将hook修复
  - ◆ 不稳定，容易crash

# 修复Zygote

- ◆ 如何保证Zygote进程不被hook?
  - ◆ Init启动Zygote进程
  - ◆ 在hook之前就将Zygote保护起来
    - ◆ 怎么保证在其他App之前启动?
  - ◆ 学习Xposed, 替换app\_process?
    - ◆ 需要重启
    - ◆ 不够稳定, 兼容性
    - ◆ 对所有App有效, 可配性不高



# App劫持修复：结论

- ◆ Hook摘除不稳定，容易crash
- ◆ 三个“基本”结论：
  - ◆ Hook基本无法修复
  - ◆ 注入基本无法剔除
  - ◆ Zygote基本无法保护



**RSA** CONFERENCE **2014**  
ASIA PACIFIC & JAPAN



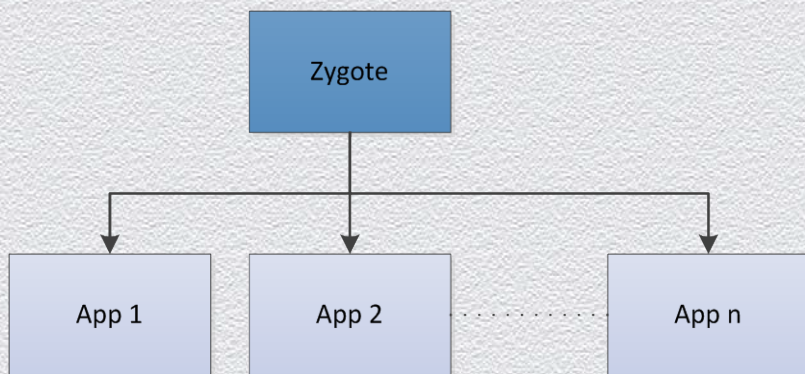
离我的App远点！——  
创建可信App运行环境

# 创建App可信运行环境

- ◆ 创建可信的Zygote进程
- ◆ 保护可信的Zygote
- ◆ 让App从干净的Zygote进程fork

# 创建可信Zygote进程

- ◆ Step1:启动一个新的Zygote进程
  - ◆ 编译自定义app\_process
  - ◆ 无需重启



# 保护可信Zygote进程

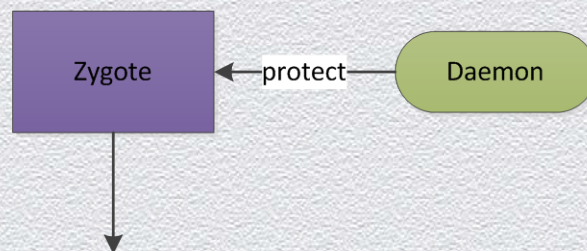
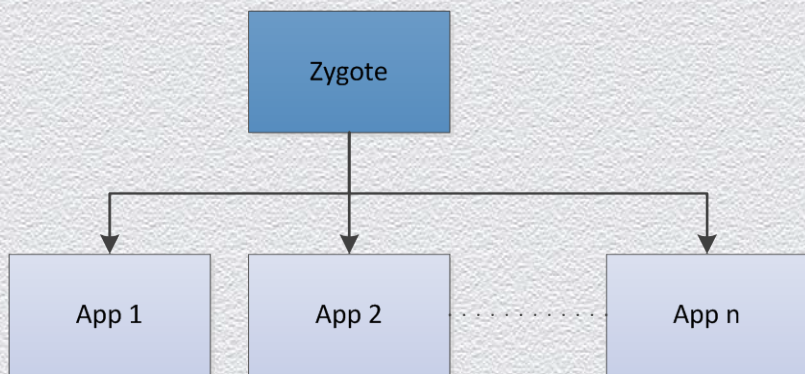
- ◆ Step2: 保护可信Zygote

- ◆ 反调试/反注入

- ◆ Ptrace\_me

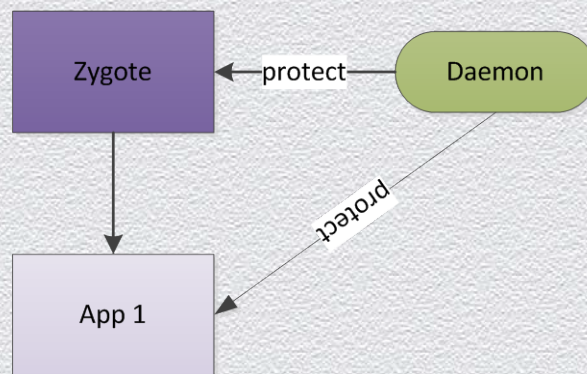
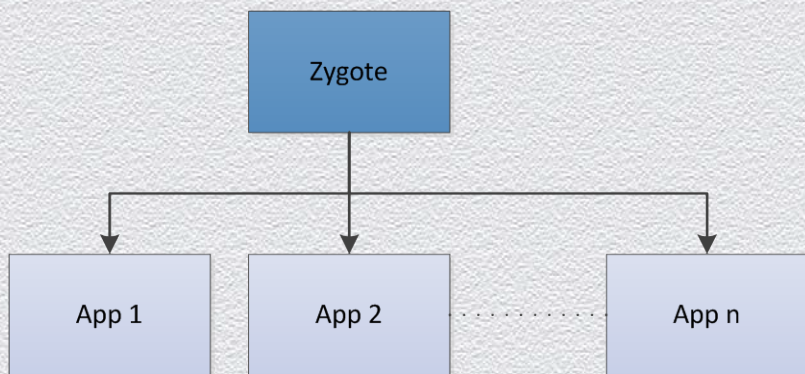
- ◆ 双进程保护

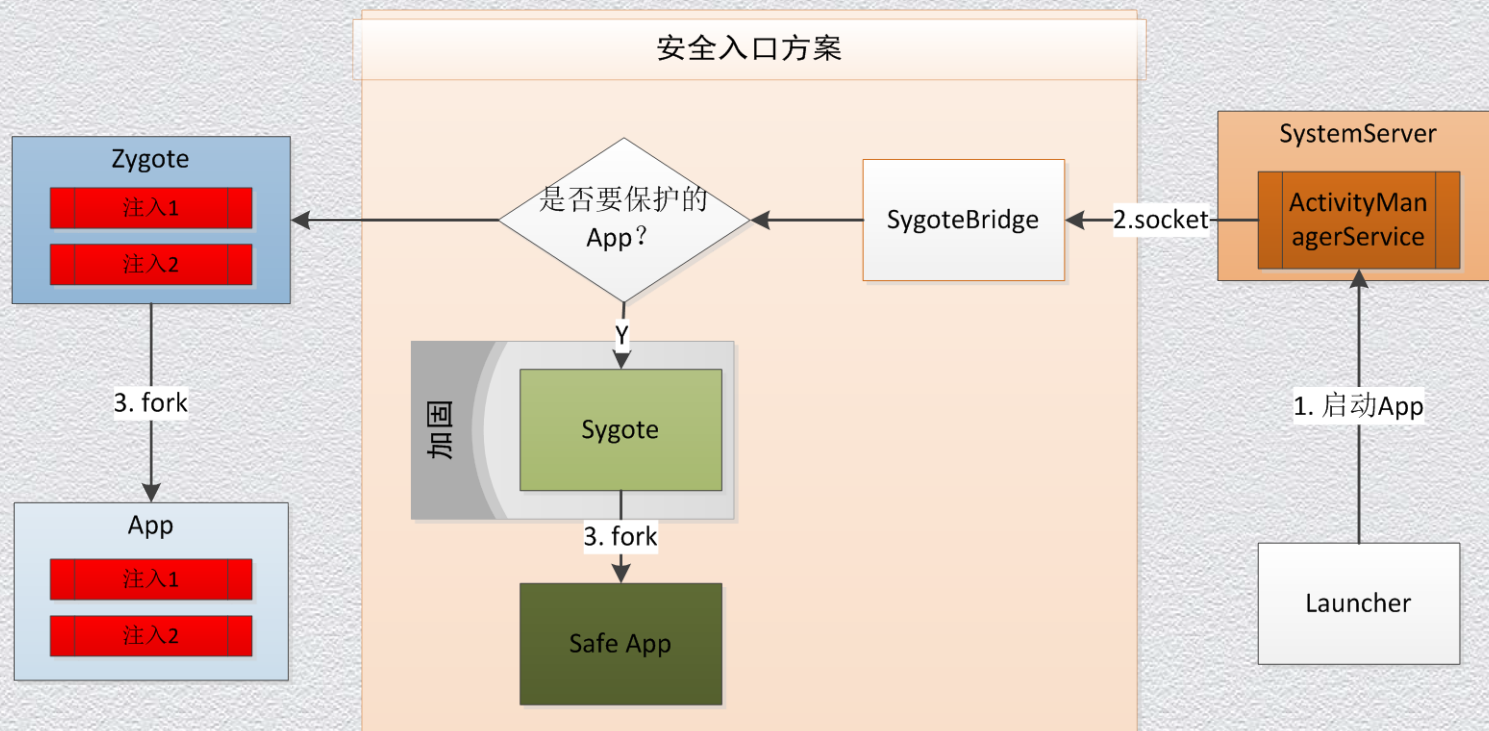
- ◆ 其他保护



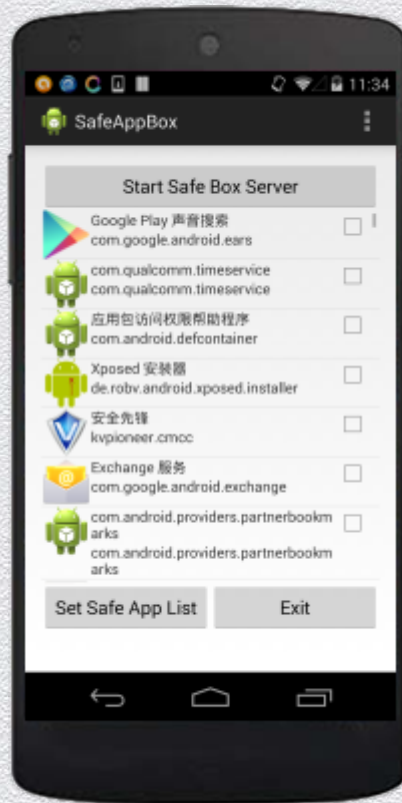
# 控制可信Zygote出口

- ◆ 通过可信Zygote fork自定义App
  - ◆ 重定向ActivityManagerService发送的socket client
  - ◆ 控制进程
    - ◆ 需要保护的App: 发送给可信Zygote
    - ◆ 无需保护的App: 发送给旧Zygote





可信运行环境：让你的App远离骚扰



**Demo:** 现在你可以决定哪些应用可以“安全地”运行在非root环境下了！



**RSAC** CONFERENCE **2014**  
ASIA PACIFIC & JAPAN



Thanks / Q&A