RSA CONFERENCE 2014
ASIA PACIFIC & JAPAN

Share.
Learn.
Secure.
Capitalizing on
Collective Intelligence

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

#RSAC

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

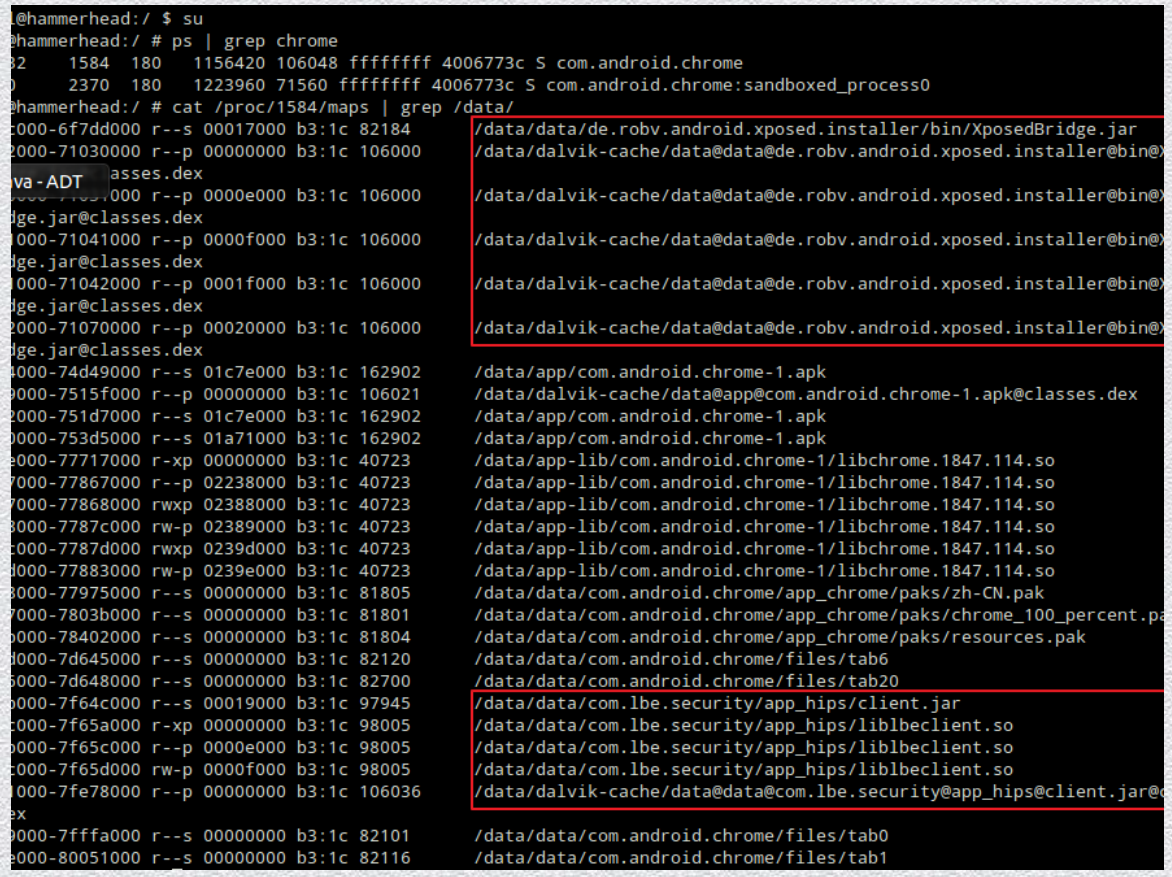
Root or Not Root

Root or Not Root

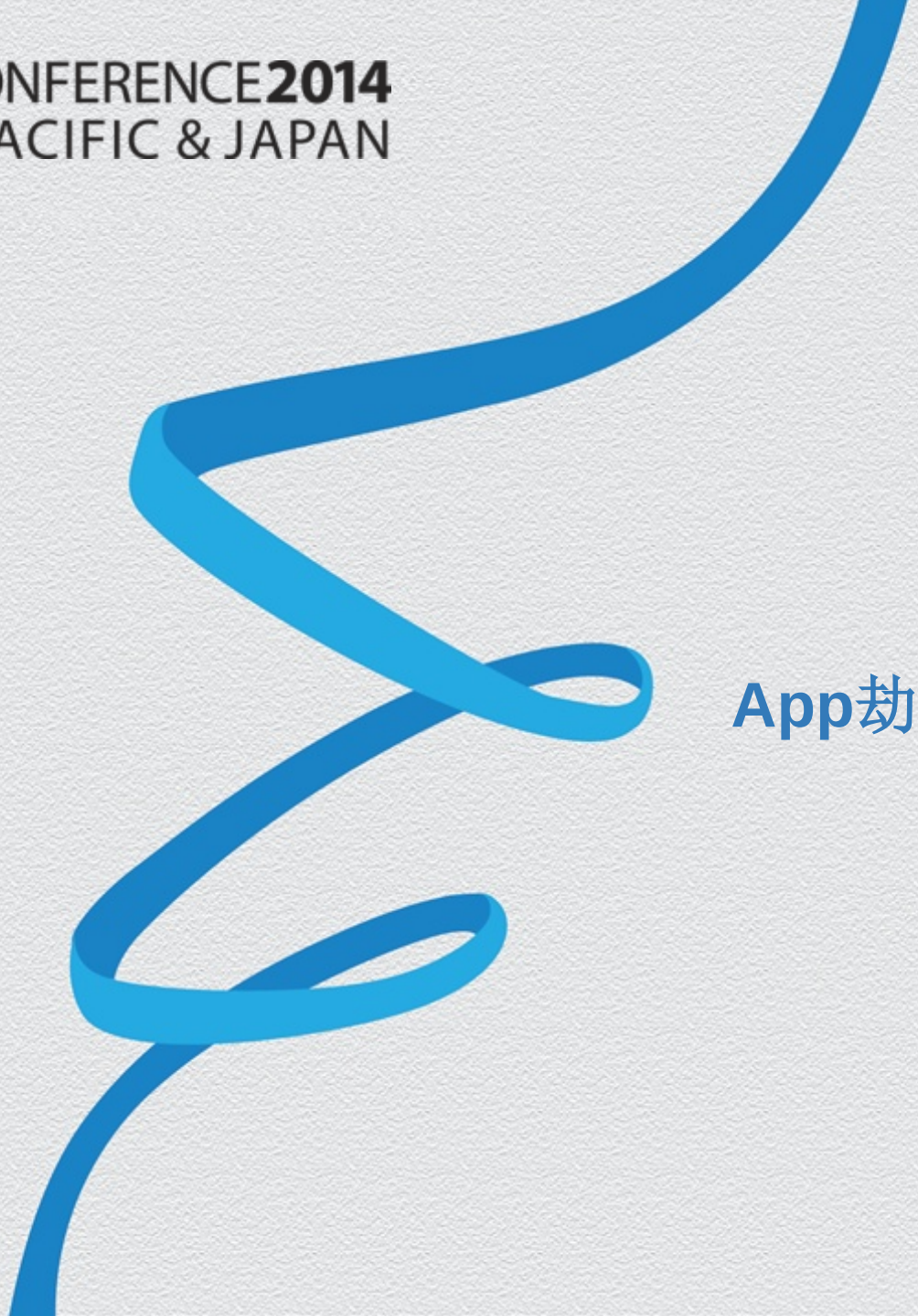
- 这不是我们今天要讨论的问题
- 不过这个问题其实很简单
  - 需要：卸载预装、控制自启动、个性化、……
  - 不需要：不安全！
- 我们需要的是“安全的Root”

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
  - ……



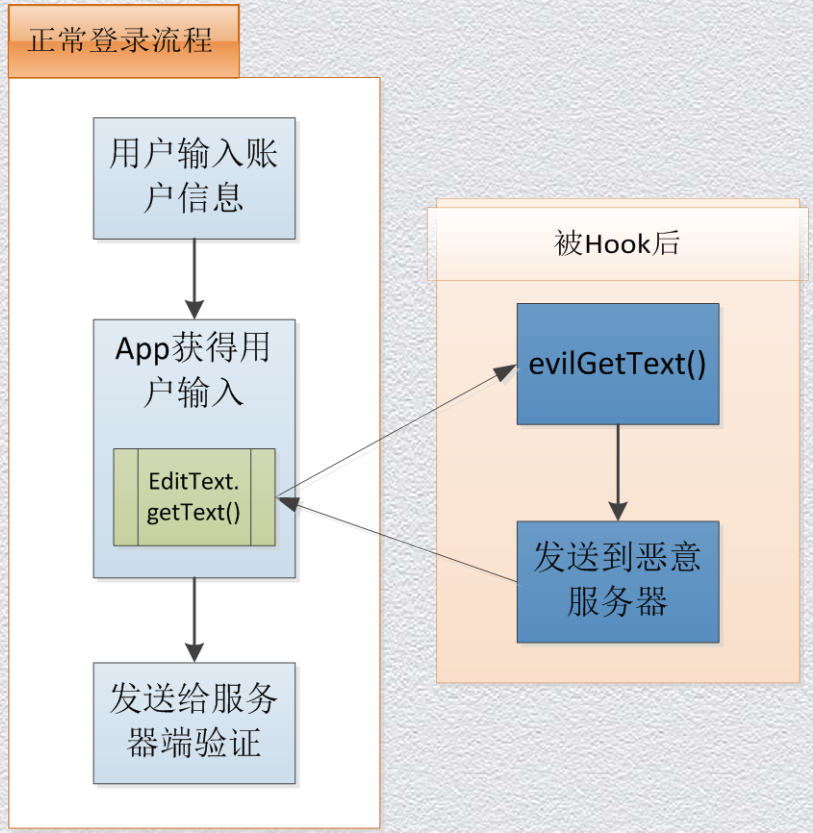
被注入后的Chrome进程



App劫持

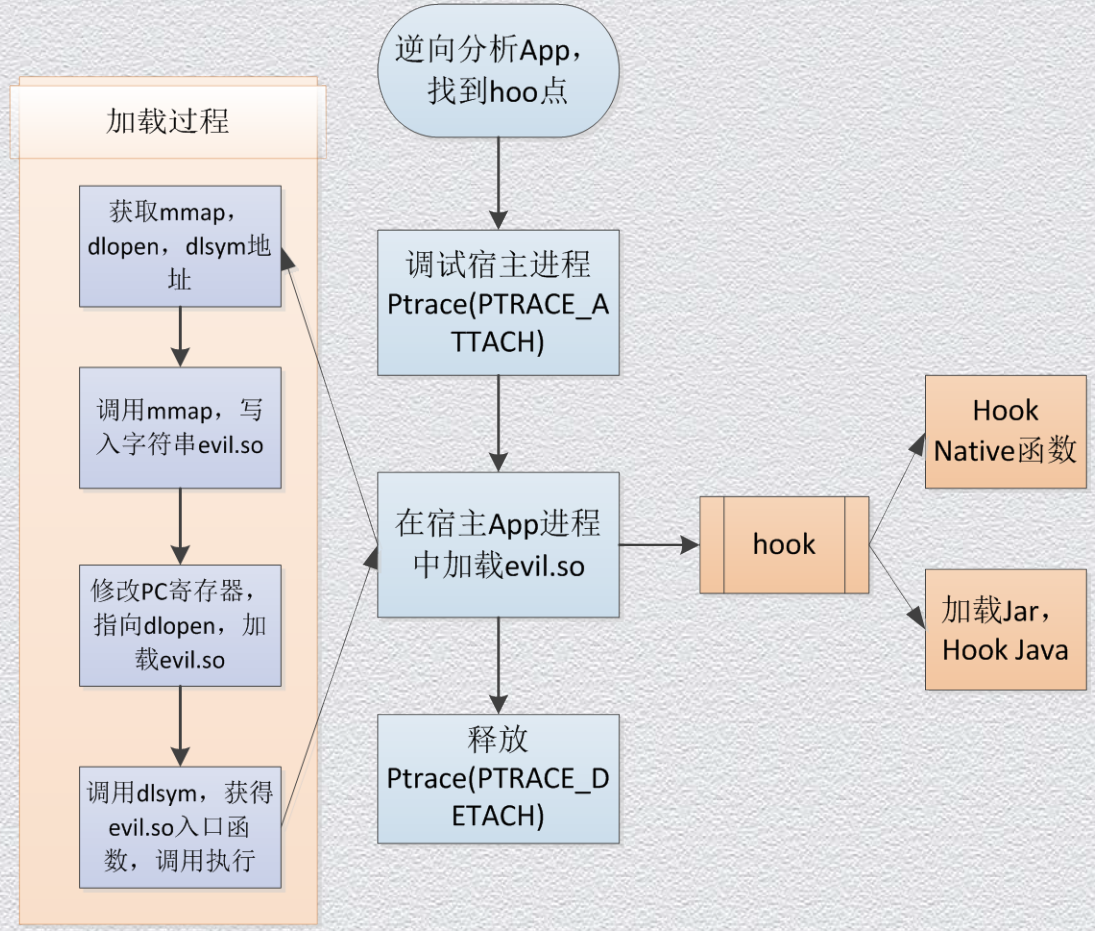
什么是App劫持?

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

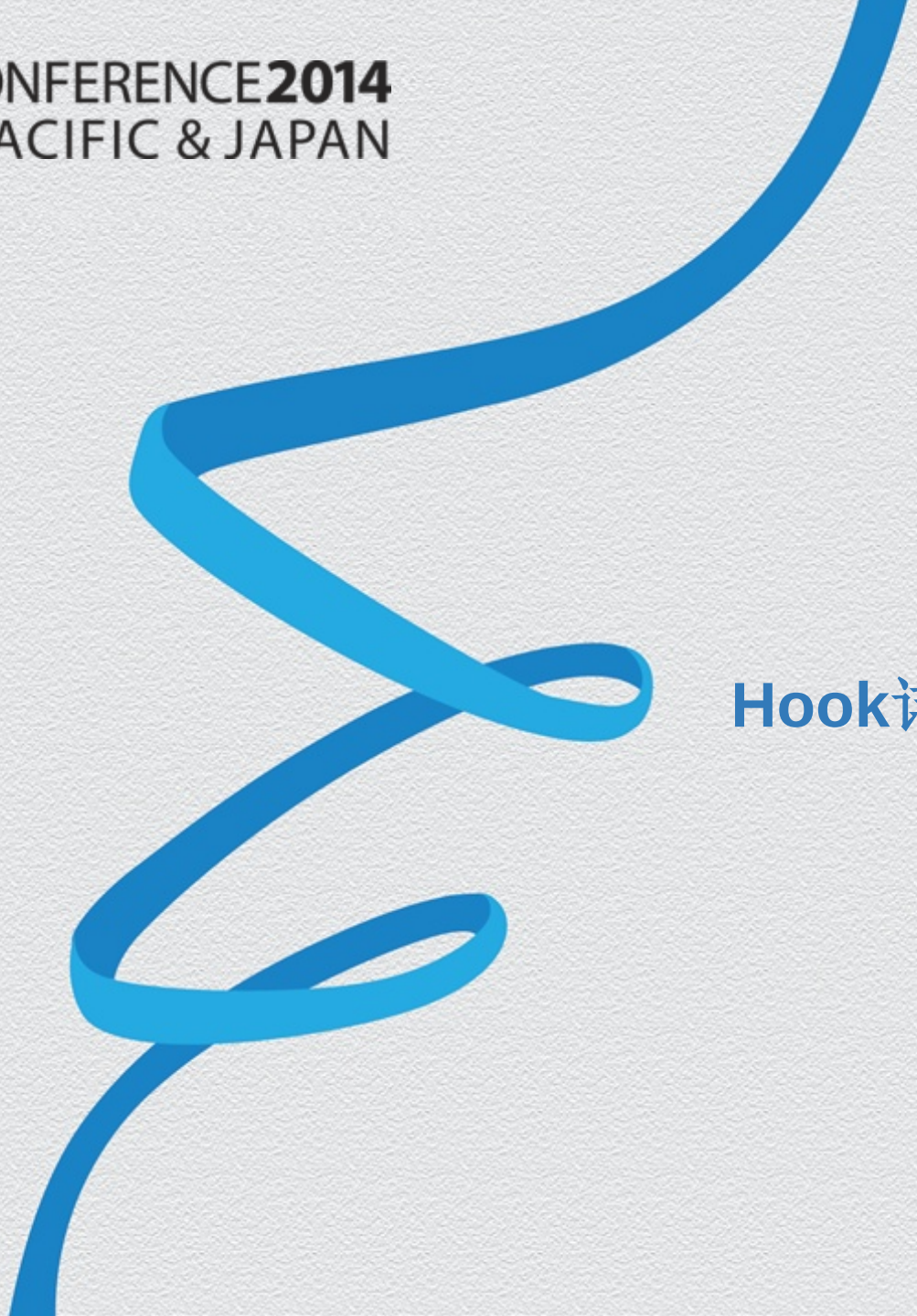


App劫持过程

- 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



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

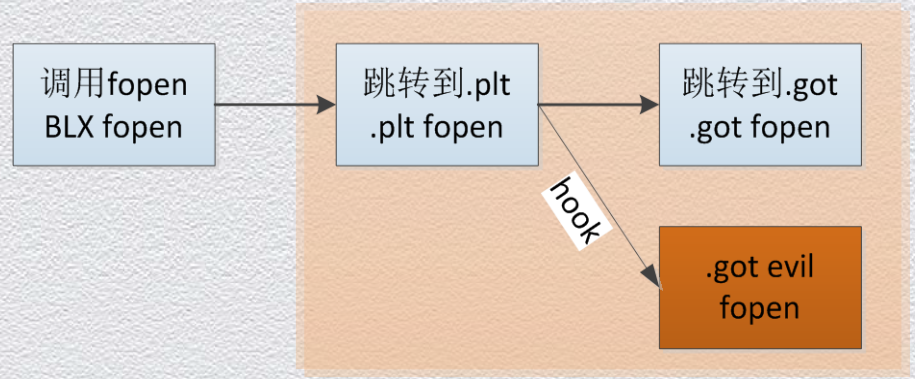
- 修改App加载的so中Got表里函数的绝对地址
- GOT
  - Global Offset Table：全局偏移表
  - 存储了依赖库so的函数的绝对地址
- PLT
  - Procedure Linkage Table：过程链接表
  - 将当前so内对依赖库函数的相对调用转移到Got中的绝对地址

So GOT Hook

- 正常执行流程

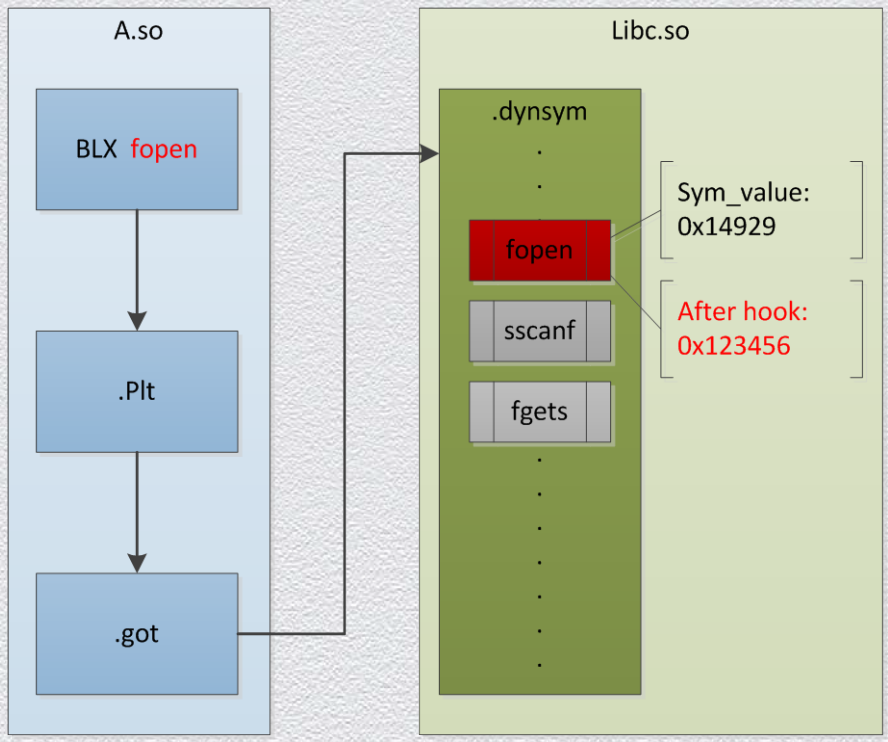
```
.text:00008C92 02 98                                    LDR     R0, [SP,#0x640+filename] ; filename
.text:00008C94 79 44                                    ADD     R1, PC          ; "w+"
.text:00008C96 FF F7 E8 ED                              BLX     fopen

.plt:00008868                       ; FILE *fopen(const char *filename, const char *modes)
.plt:00008868                       fopen                                       ; 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

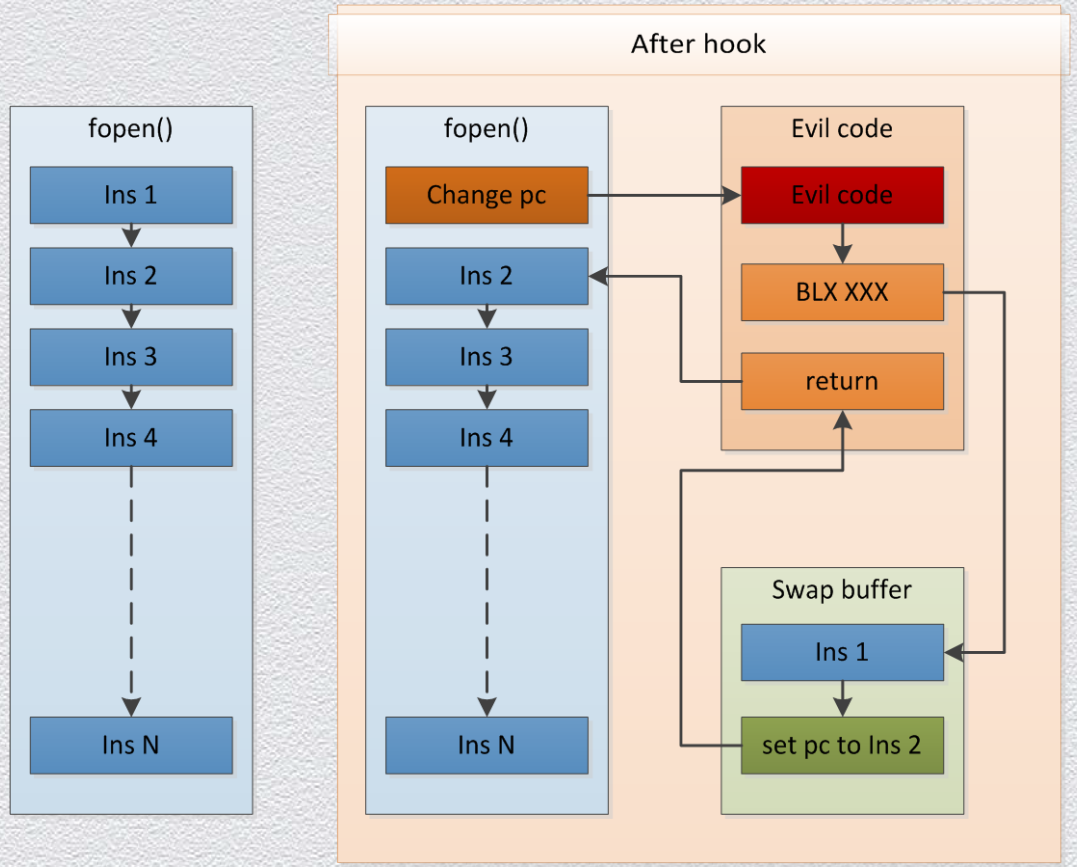
- 修改App加载的So的符号表里的函数地址
- 缺点：
  - 必须在so加载前hook



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



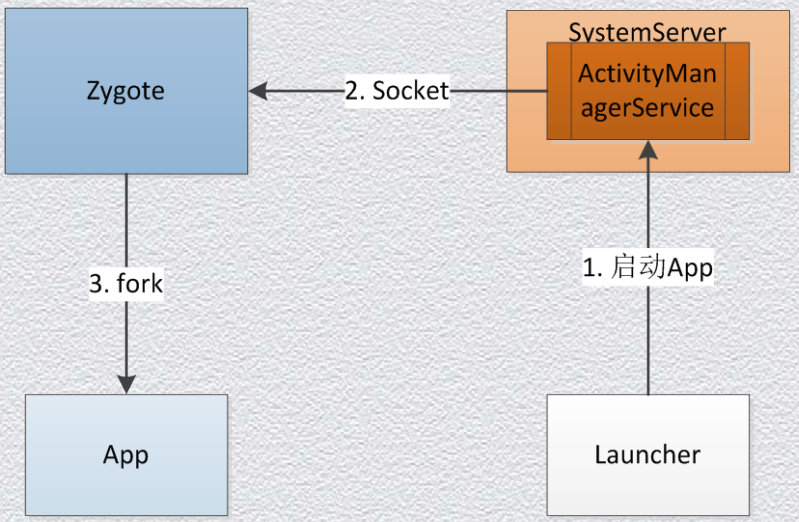
So inline Hook流程

Hook Zygote

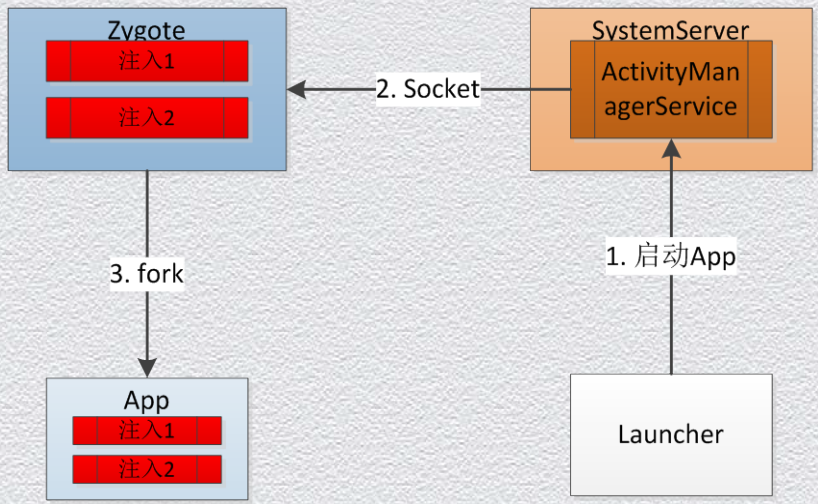
- 全局hook的方法：
  - Hook当前已运行的所有App进程，监听新进程
  - Hook Zygote进程
- App进程是有Zygote孵化出来的
  - 如果Zygote被注入/hook，其他App都将被注入/hook

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

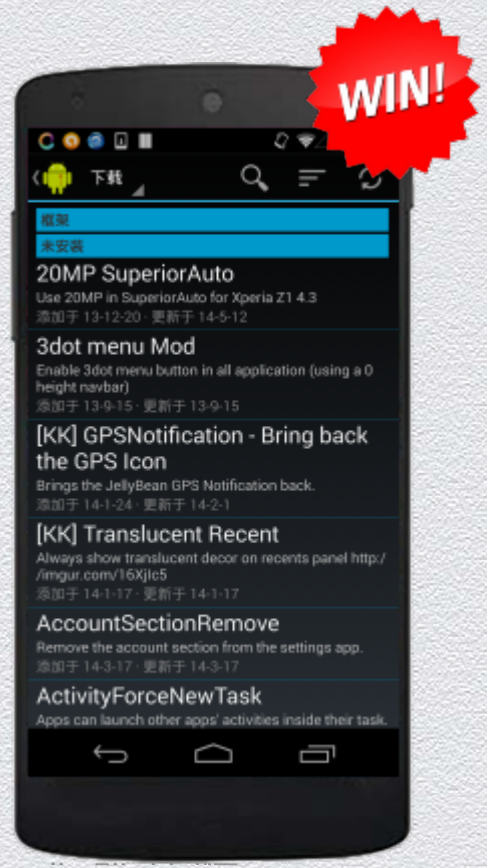
- 目前使用最广的Android平台开源hook框架
- 替换App_Process
- Java函数转为Native函数
- 在被hook函数执行时，调用BeforeHook/AfterHook接口
- 添加/删除hook需要重启zygote

流行的Hook框架：Cydia Substrate

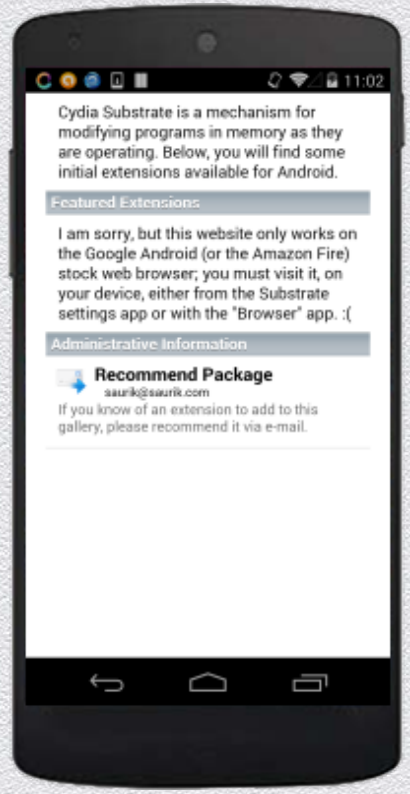
- 兼容iOS和Android
- 在iOS平台使用广泛
- 需要重启Zygote

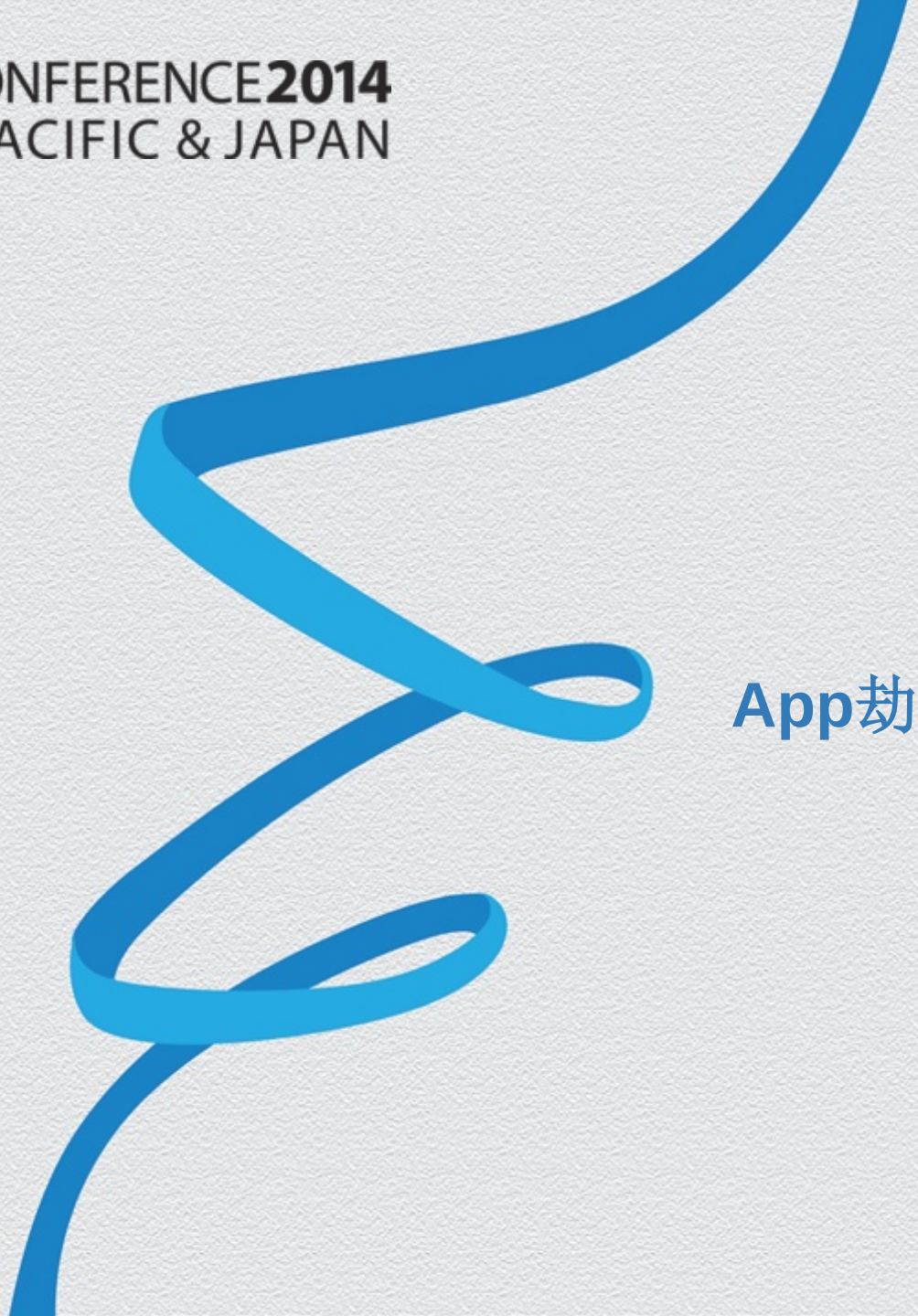
流行的Hook框架对比

XPosed



Cydia Substrate

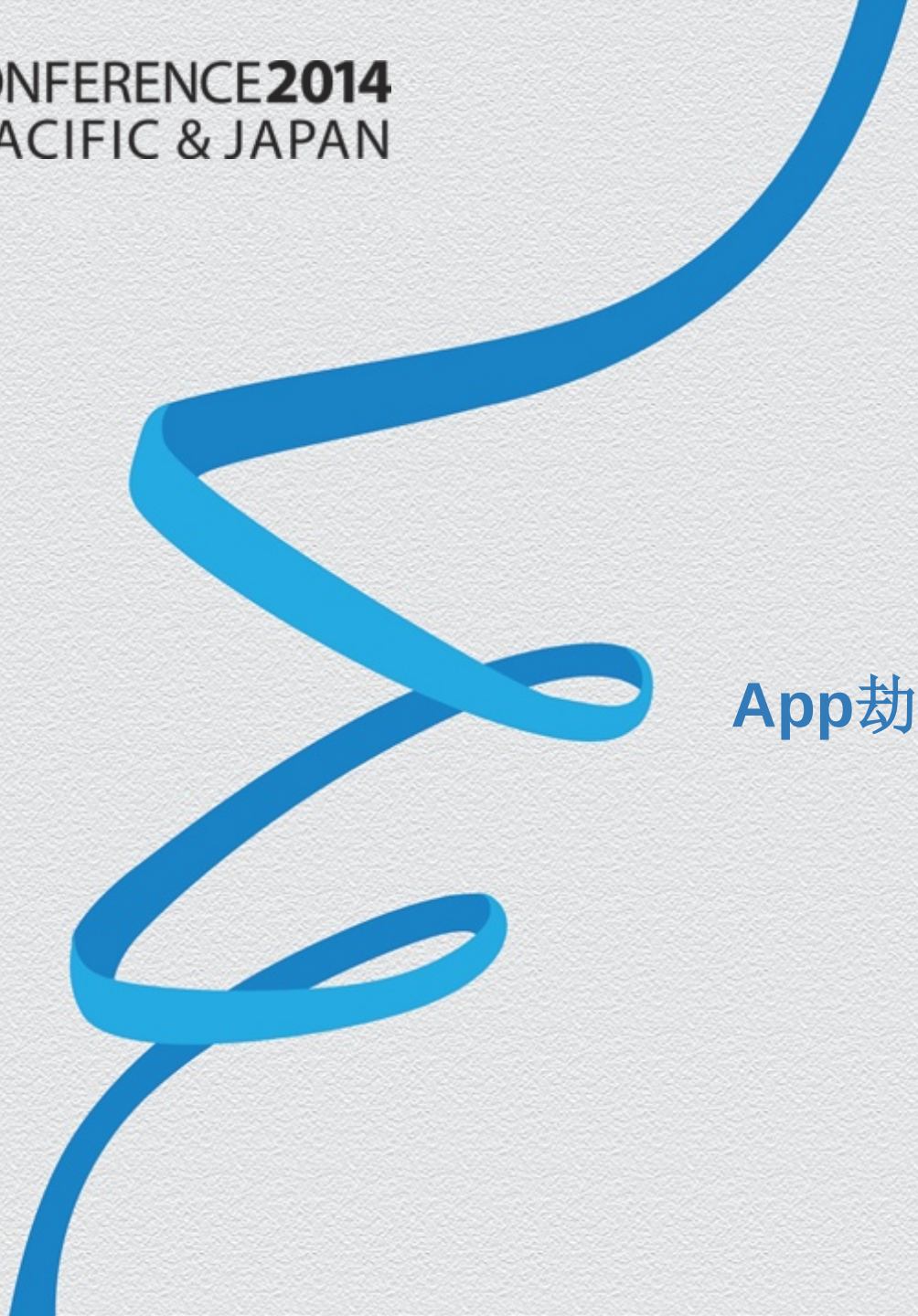




App劫持演示

Demo：Facebook登陆劫持

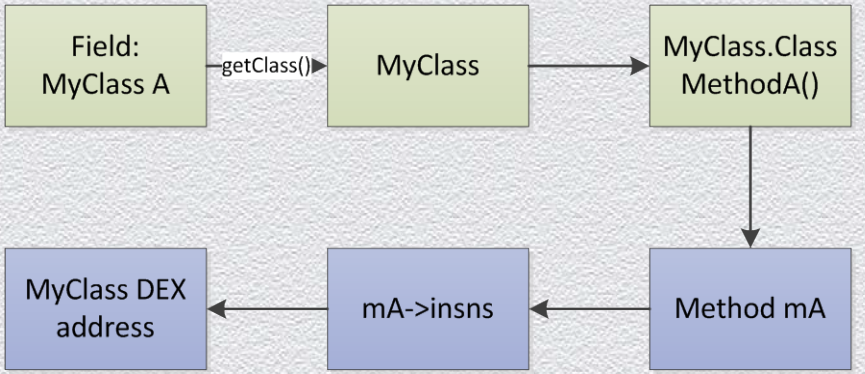
Demo：招行网银WebView钓鱼



App劫持检测/修复

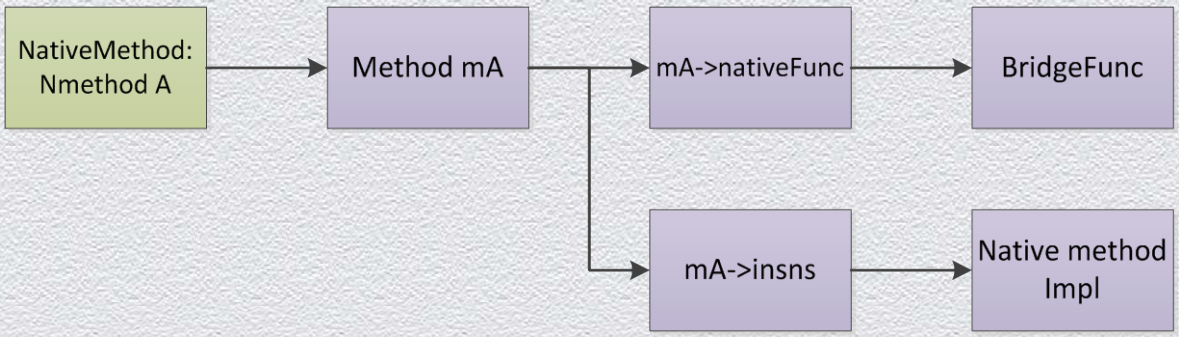
Hook检测/修复：Java Hook

- 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

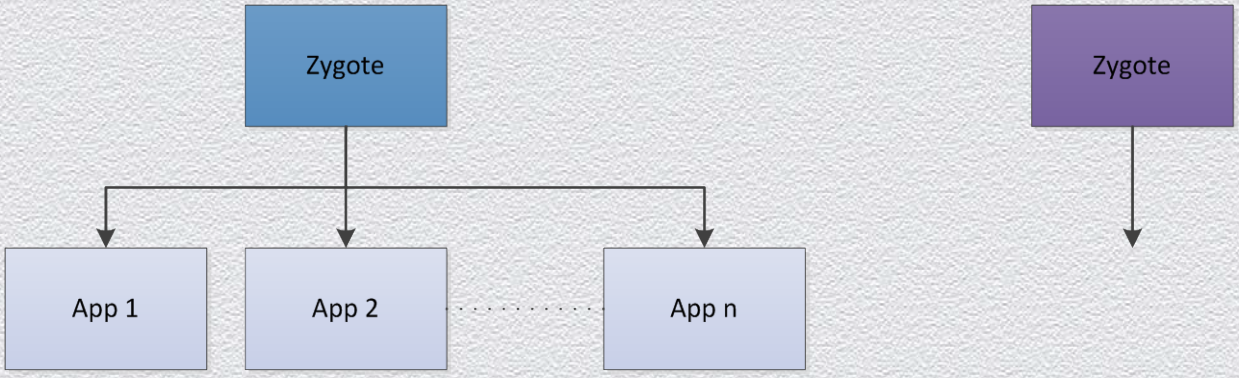
#RSAC

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

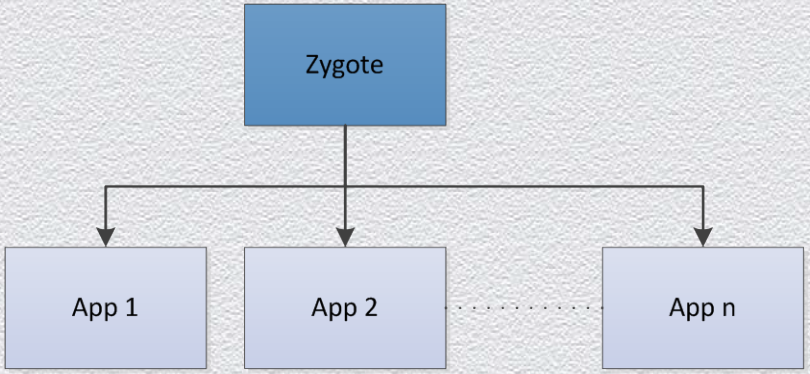
创建可信Zygote进程

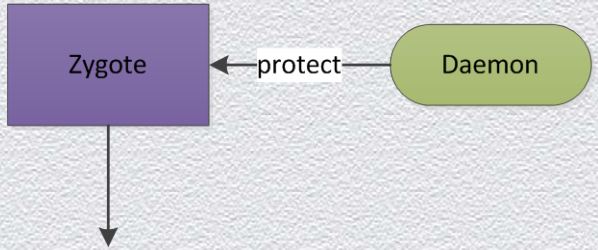
- Step1:启动一个新的Zygote进程
  - 编译自定义app_process
  - 无需重启



保护可信Zygote进程

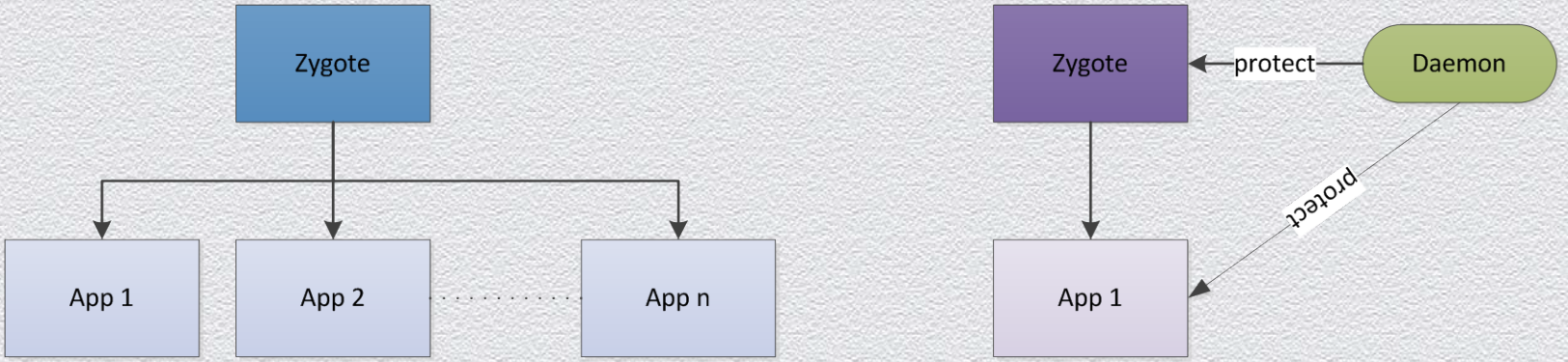
- Step2: 保护可信Zygote
  - 反调试/反注入
    - Ptrace_me
    - 双进程保护
  - 其他保护

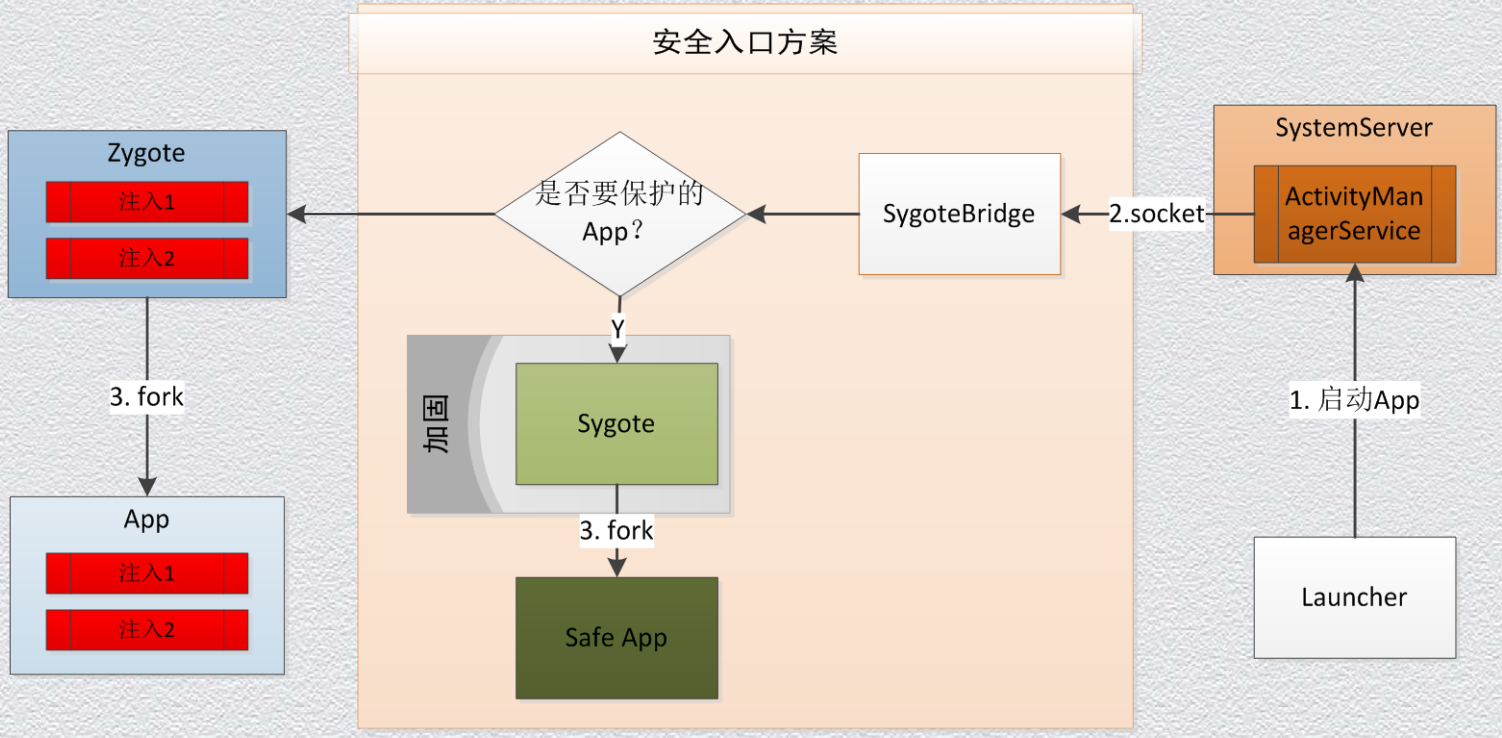




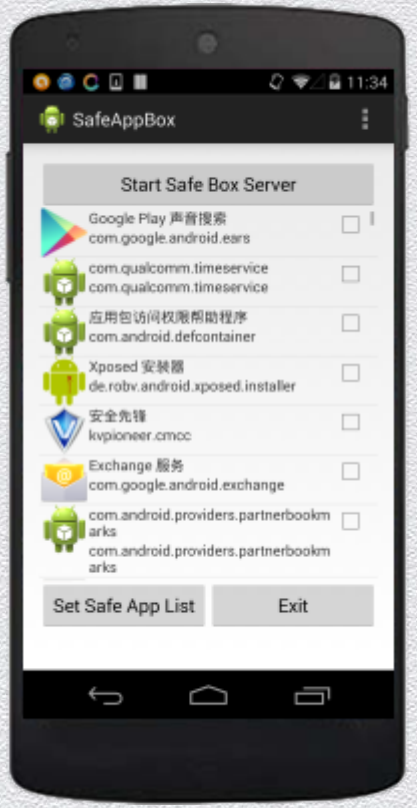
控制可信Zygote出口

- 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

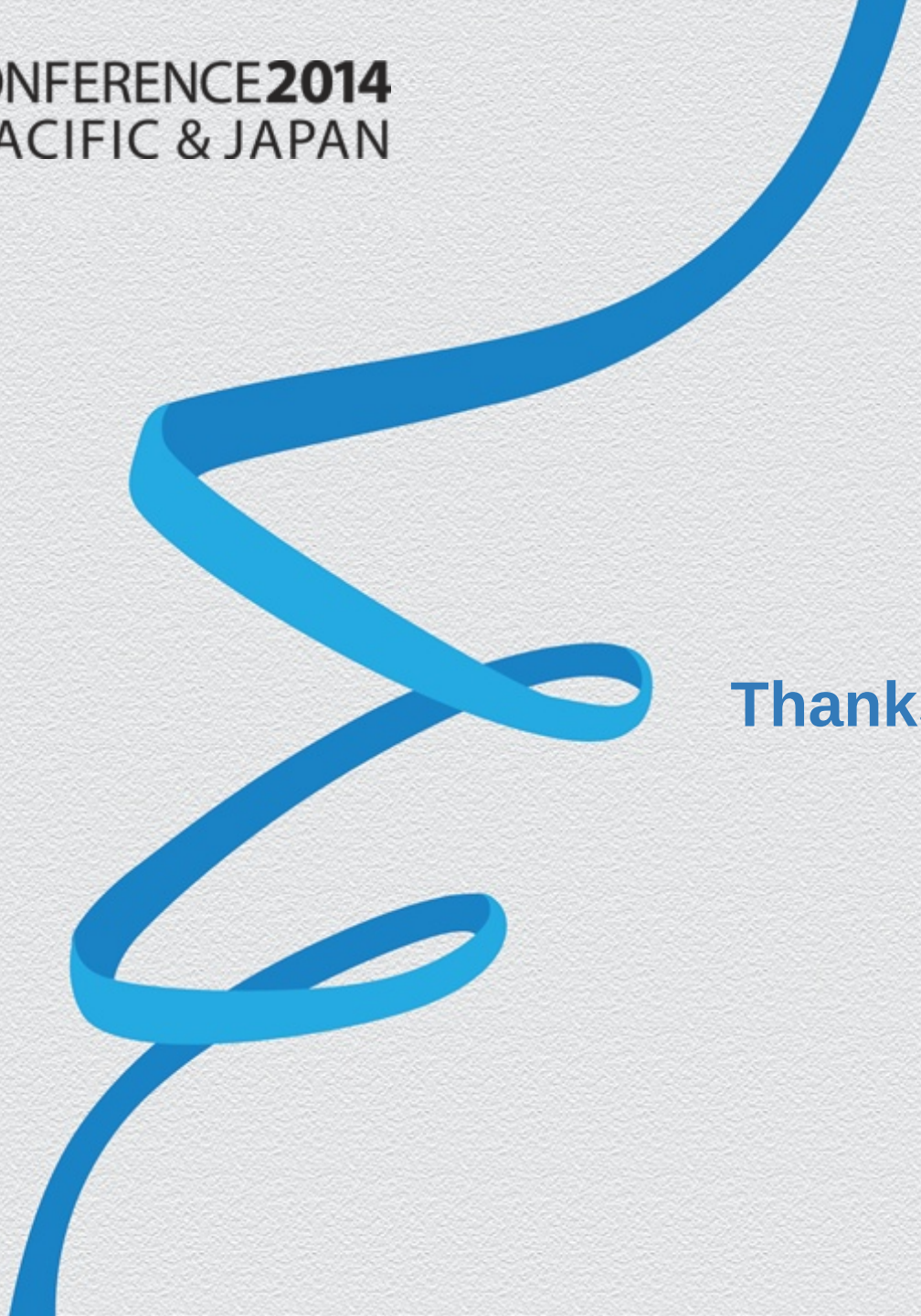




可信运行环境：让你的App远离骚扰



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！



Thanks / Q&A