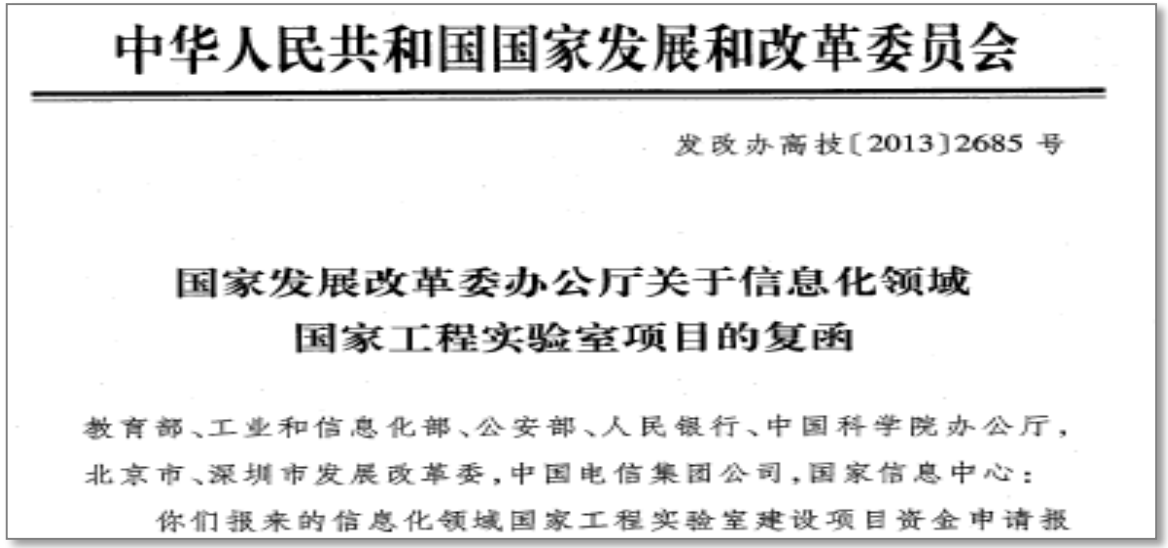
基于真实Android环境下的APP程序分析与安全检测

移动互联网系统与应用安全国家工程实验室

2015年1月

关于实验室：发改委批复

- 建设任务：针对移动互联网安全热点、难点，围绕新需求，建设技术研发和工程化平台，开展关键技术、设备的研发、集成和工程化。
- 建设目标：
  1. 未来3年突破8项关键技术研究和设备开发；
  2. 完善相关仿真测试平台和试验验证系统；
  3. 建成可信应用软件验证和检测平台，自动化扫描、检测达90%以上；
  4. 支付业务风险人工调查流程减少80%以上；
  5. 申请专利不少于10项；推进相关技术标准修订；
  6. 为推进技术进步、产业发展提供技术支撑。
- 建设内容：建设3个研发平台和相关试验系统。



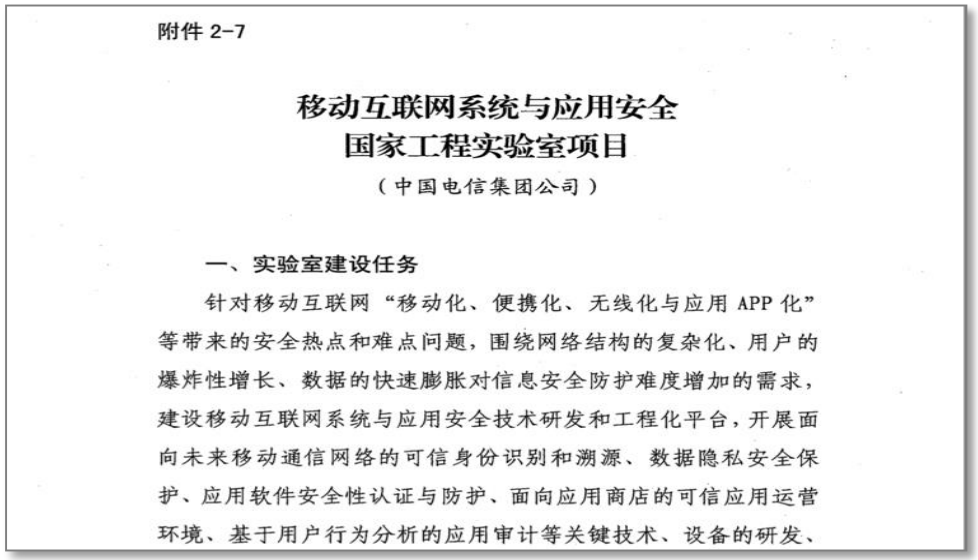
中华人民共和国国家发展和改革委员会

发改办高技〔2013〕2685号

国家发展改革委办公厅关于信息化领域国家工程实验室项目的复函

教育部、工业和信息化部、公安部、人民银行、中国科学院办公厅，北京市、深圳市发展改革委，中国电信集团公司，国家信息中心：

你们报来的信息化领域国家工程实验室建设项目资金申请报



附件 2-7

移动互联网系统与应用安全国家工程实验室项目

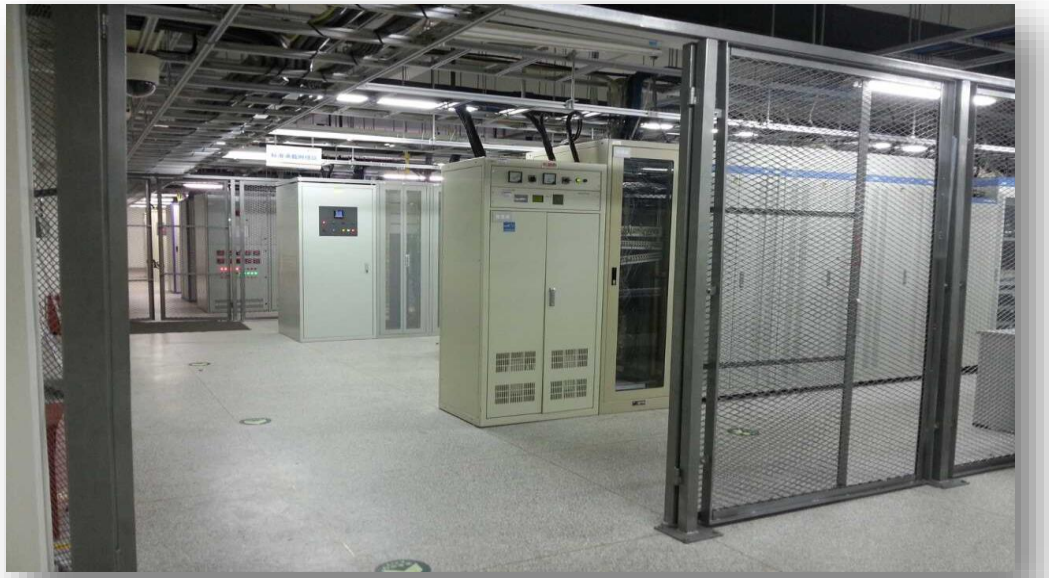
（中国电信集团公司）

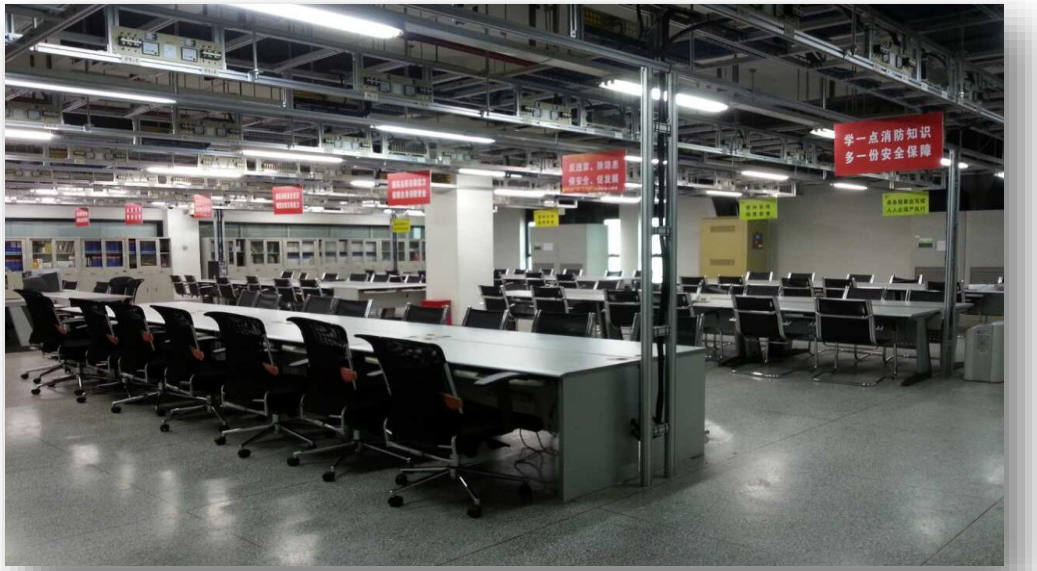
一、实验室建设任务

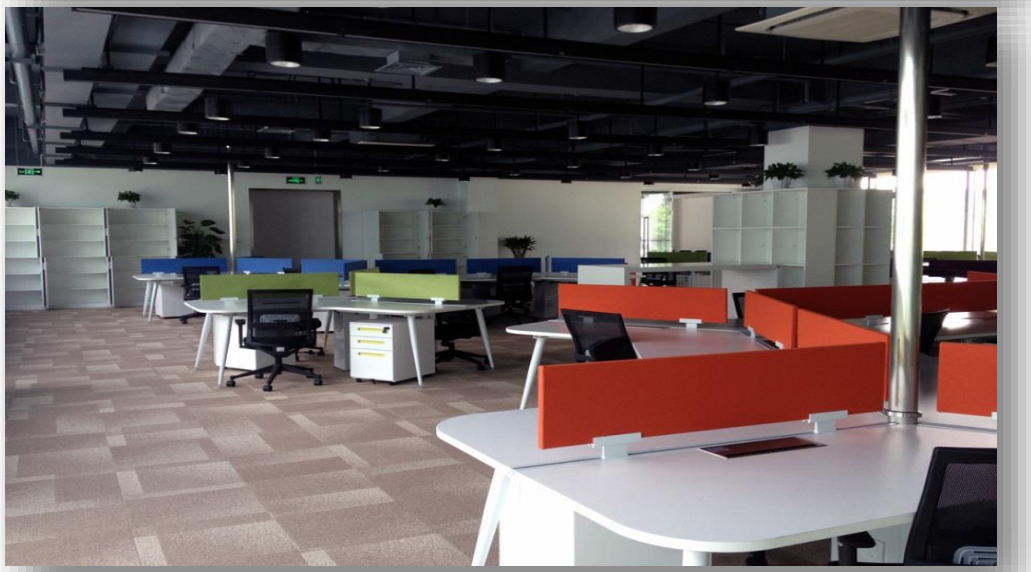
针对移动互联网“移动化、便携化、无线化与应用 APP 化”等带来的安全热点和难点问题，围绕网络结构的复杂化、用户的爆炸性增长、数据的快速膨胀对信息安全防护难度增加的需求，建设移动互联网系统与应用安全技术研发和工程化平台，开展面向未来移动通信网络的可信身份识别和溯源、数据隐私安全保护、应用软件安全性认证与防护、面向应用商店的可信应用运营环境、基于用户行为分析的应用审计等关键技术、设备的研发、

2

关于实验室：工作环境

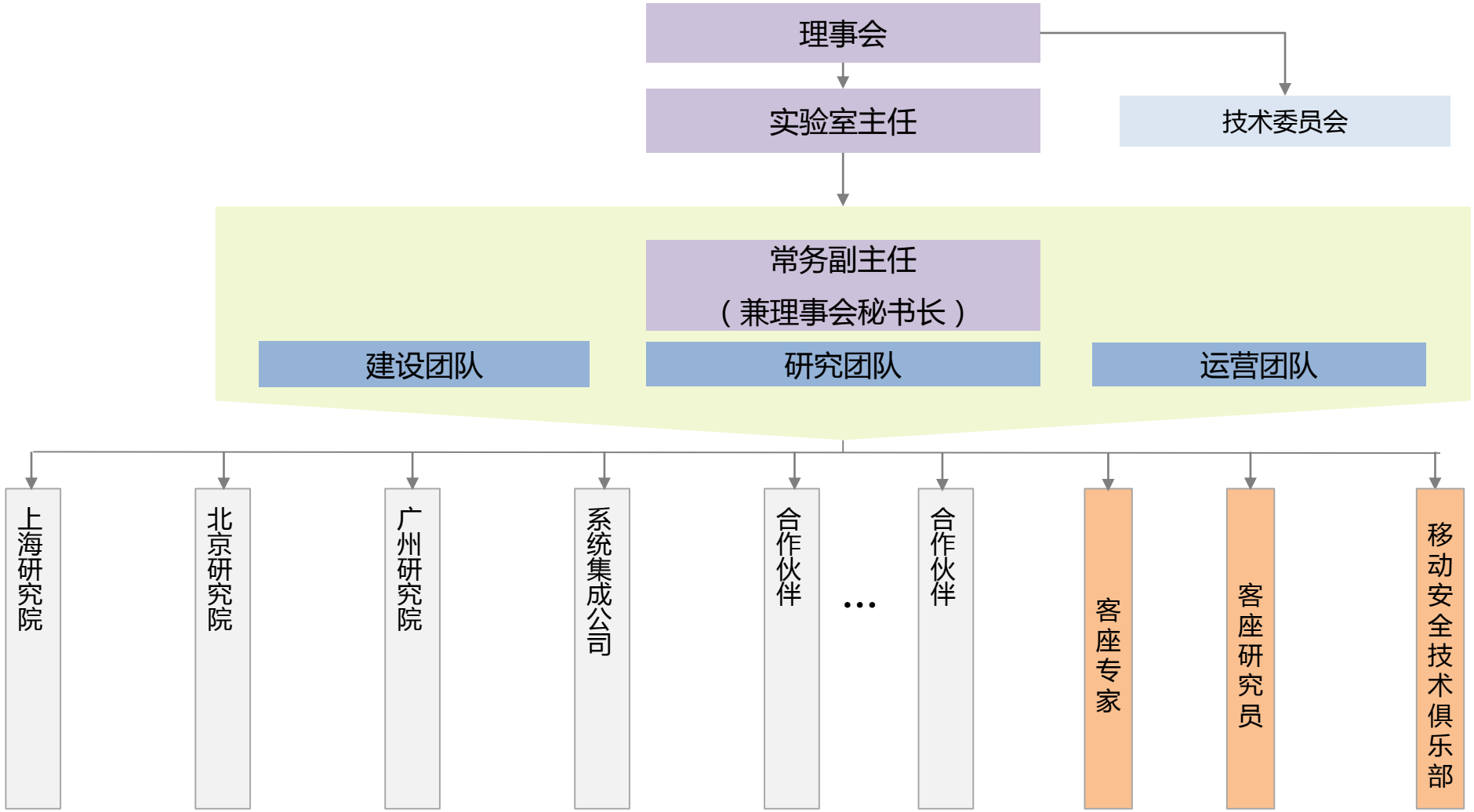






3

关于实验室：组织架构


理事会
实验室主任
技术委员会
常务副主任
（兼理事会秘书长）
建设团队
研究团队
运营团队
上海研究院
北京研究院
广州研究院
系统集成公司
合作伙伴
...
合作伙伴
客座专家
客座研究员
移动安全技术俱乐部

主讲人介绍

- 报告人：**杨文博**
  - **上海掌御信息科技有限公司高级技术工程师**
  - 上海交通大学计算机科学与应用专业博士
- 个人经历：
  - 专注于Android移动应用APP安全分析
  - 设计与开发了动态APP分析引擎**InDroid**
  - 承担国家科技重大专项、上海市科委专项等移动安全研究项目
  - 承担过银联、华为等企业级移动安全审查工作
  - 2015移动安全挑战赛（阿里巴巴、看雪主办）**第一名**（超过一千名国内移动安全研究人员参赛）
  - 2014各项信息安全竞赛APK分析类问题解答累积最多
  - 2012年Honeynet Mobile Forensic挑战赛**全球亚军**

5

合作伙伴：上海掌御信息科技有限公司

- **上海掌御信息科技有限公司**
  - 移动互联网系统与应用安全国家工程实验室**技术合作伙伴**
  - 移动安全技术俱乐部成员

- 公司背景：
  - Android移动应用APP安全分析服务、系统安全加固服务
- 主要产品
  - Android移动安全检测平台
  - Android数据加固平台
  - Android隐私保护加固系统
  - Android移动办公安全加固SDK

6

Content

Part 1

Part 2

Part 3

Part 4

APP分析的现状与工具

基于模拟器分析的不足之处

APP高级程序分析需求

基于真实环境的APP分析

APP分析

- 静态分析
  - —APK反汇编/反编译
  - —程序分析

- 动态分析
  - —调试
  - —关键函数Hook
  - —系统事件监视

现有分析工具一览

反汇编/编译	动态分析	程序分析	相似性检测	Sandbox
Dexdump	Andbug	FlowDroid	DNADroid	DroidBox
Smali	GikDbg	AManDroid	Juxtapp	Anubis
Dexter	IDA 6.6	AndroGuard	DroidMOSS	SandDroid
JD-GUI	Aurasium	TaintDroid	ViewDroid	CopperDroid
JAD	Drozer	WoodPecker	PlayDrone	Genymotion
SOOT	Xposed	IntentFuzzer	Centroid	PreCrime
AndroGuard	NDroid	CryptoLint	PiggyApp	TraceDroid

- 参考 http://wiki.secmobi.com/

基于Sandbox的动态分析

- Sandbox
  - ——一类重要的分析工具

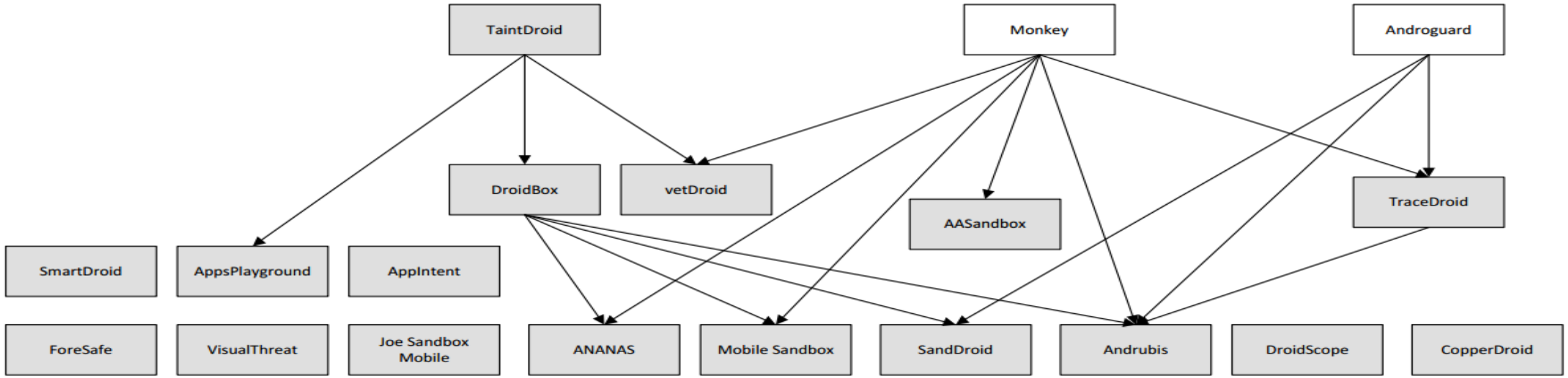
- 动态分析
  - —对抗代码加壳、混淆等
  - —监控各类事件

沙盒系统需求

- 设计一个沙盒分析系统
  - —APP静态分析
  - —APP动态分析
  - —事后离线分析

- 特征
  - —快速复原初始状态
  - —自动化测试
  - —全面监控

沙盒系统比较


TaintDroid
Monkey
Androguard
DroidBox
vetDroid
AASandbox
TraceDroid
SmartDroid
AppsPlayground
AppIntent
ForeSafe
VisualThreat
Joe Sandbox Mobile
ANANAS
Mobile Sandbox
SandDroid
Andrubis
DroidScope
CopperDroid

模拟器的作用

- 大部分Sandbox系统采用了Emulator
  - —全系统分析（特别是ARM Native code）

- 方便动态分析
  - —支持native级别的调试
  - —在桌面计算机系统或服务器上运行
  - —并发

Content

Part 1 APP分析的现状与工具

Part 2 基于模拟器分析的不足之处

Part 3 APP高级程序分析需求

Part 4 基于真实环境的APP分析

模拟器 vs 真实设备

■适合大规模粗粒度自动化安全分析

- •经济成本低
- •高度可定制
- •容易部署

■适合小规模细粒度人工深入分析

- •相对比较昂贵
- •修改系统（ROM）需要一定条件
- •难以大规模并行

模拟器检测：永恒的武器竞赛

- 模拟分析系统特征的检测
  - —用户层行为和数据
  - —Android系统层特征
  - —Linux系统层特征
  - —模拟器体系结构特征

模拟器检测：永恒的武器竞赛

- 对抗沙盒系统，寻找沙盒系统的Fingerprint
  - —静态特征：ID、特定文件
  - —动态特征：硬件反馈（GPS，陀螺仪）
  - —硬件Performance：CPU、GPU

- 事实上，模拟器差异很难消除
  - —Testing CPU emulators @ISSTA 2009
  - —仔细检查CPU指令集中部分指令实现的不一致性

模拟器检测：永恒的武器竞赛

- 隐蔽自身
  - —修改Emulator配置
  - —提供精确的硬件事件模拟
  - —结合真实设备进行模拟

- 检测APP的“检测过程”
  - —参考：胡文君、肖梓航. Guess Where I am: Android模拟器躲避的检测与应对. Hitcon 2014

Content

Part 1

Part 2

Part 3

Part 4

APP分析的现状与工具

基于模拟器分析的不足之处

APP高级程序分析需求

基于真实环境的APP分析

大规模分析 vs 人工分析

- 越来越多的学术论文
  - —动辄分析10万甚至百万数量级的app
  - —结果的可比较性？

- 大规模分析的意义
  - —利用机器学习进行分类
  - —恶意软件相似性检测

大规模分析 vs 人工分析

- 人工分析在哪些方面依然做得更好
  - —复杂事件触发
  - —算法和协议恢复
  - —高级漏洞分析

复杂程序分析需求

- 程序行为理解
- 反混淆与反保护
- 程序验证
- 密码学算法与协议分析

程序分析目的

- 从特征识别到程序理解
  - —更好地理解一个APP的算法、协议、功能

- 困难
  - —商业软件和恶意软件都更为重视软件保护
  - —APP保护方案愈发成熟
  - —更多的方案针对已有的工具

Content

Part 1

Part 2

Part 3

Part 4

APP分析的现状与工具

基于模拟器分析的不足之处

APP高级程序分析需求

基于真实环境的APP分析

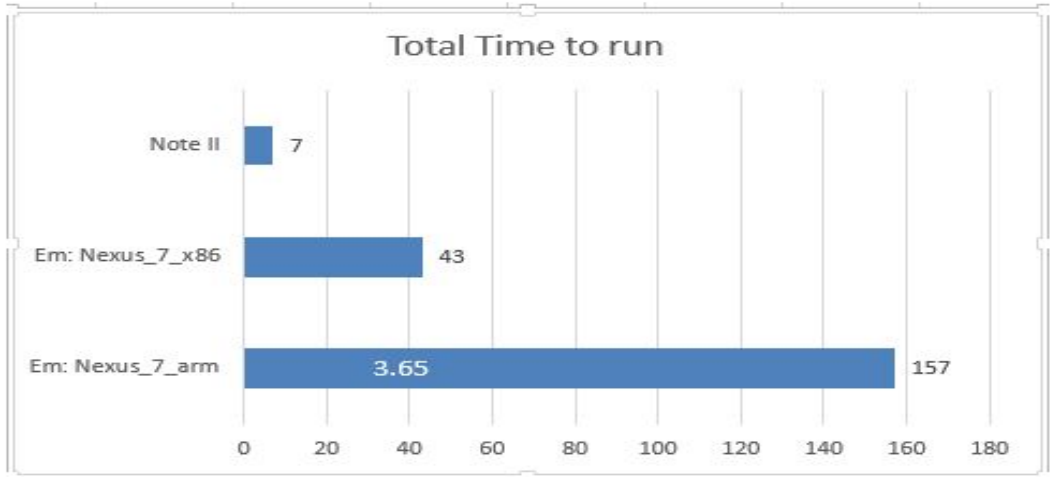
真实环境能带来哪些好处

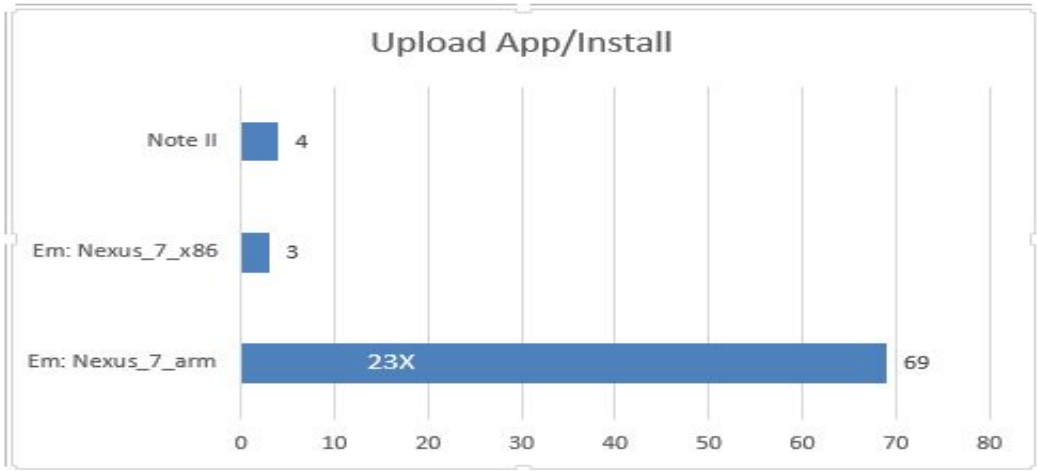
- 模拟器性能问题
  - —Android Emulator性能非常的慢
  - —原因：基于QEMU，在 x86 架构上模拟 ARM 指令集

- 加速
  - —Genymotion
  - —Intel x86 Emulator
  - —Or 使用手机

性能比较

- 普通模拟器，Intel x86下模拟器与Note2比较
  - —数据来源：https://software.intel.com/en-us/android/blogs/2013/12/11/performance-results-for-android-emulators-with-and-without-intel-haxm





真实环境能带来哪些好处

- 模拟器仿真性问题
  - —基于硬件的I/O事件无法精确模拟
  - —操作方式（鼠标vs触摸屏）

- 更好的测试

真实环境能带来哪些好处

- 真实环境与模拟器协同分析
  - —Usenix14：BareCloud: Bare-metal Analysis-based Evasive Malware Detection

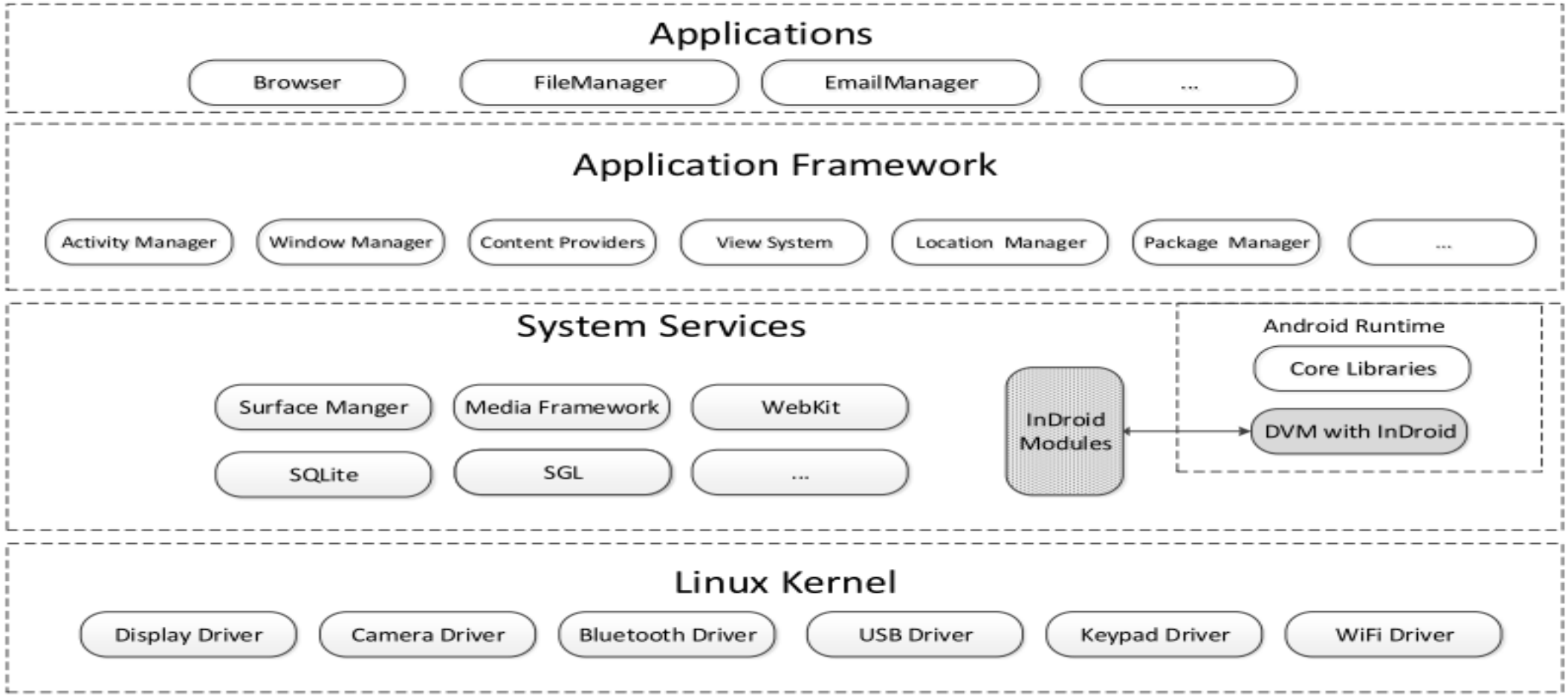
- 可利用crowdsourcing
  - —让更多实际用户参与到分析工作中来

设计一个基于真实环境的分析系统

- 基于真实Android设备
  - —可自行修改系统
  - —Bootloader解锁
  - —支持自定义recovery

- 如何打造这样的设备
  - —Nexus系列
  - —兼容AOSP的设备

设计一个基于真实环境的分析系统



设计一个基于真实环境的分析系统

- Galaxy Nexus
  - —淘宝价格600-800元
  - —良好的官方Bootloader解锁
  - —支持自定义recovery
  - —其它Nexus系统手机/平板也都非常适合

- 利用已有设备
  - —Samsung S3，S4，Note2等
  - —SONY LT28H
  - —华为Ascend D1

Dalvik Instrumentation

- 已有的程序分析监控
  - —App rewritting
  - —Method hooking
  - —Permission management
  - —IPC control

- 对于Dalvik bytecode的分析尚不成熟
  - —IDA pro 6.5以后支持调试
  - —不像x86平台拥有PIN、DynamoRio等插桩工具支持

Dalvik Instrumentation

- 改造Dalvik VM
  - —思路，为DVM提供类似JVM Tool Interface的接口
  - —支持bytecode级别指令级插桩监控

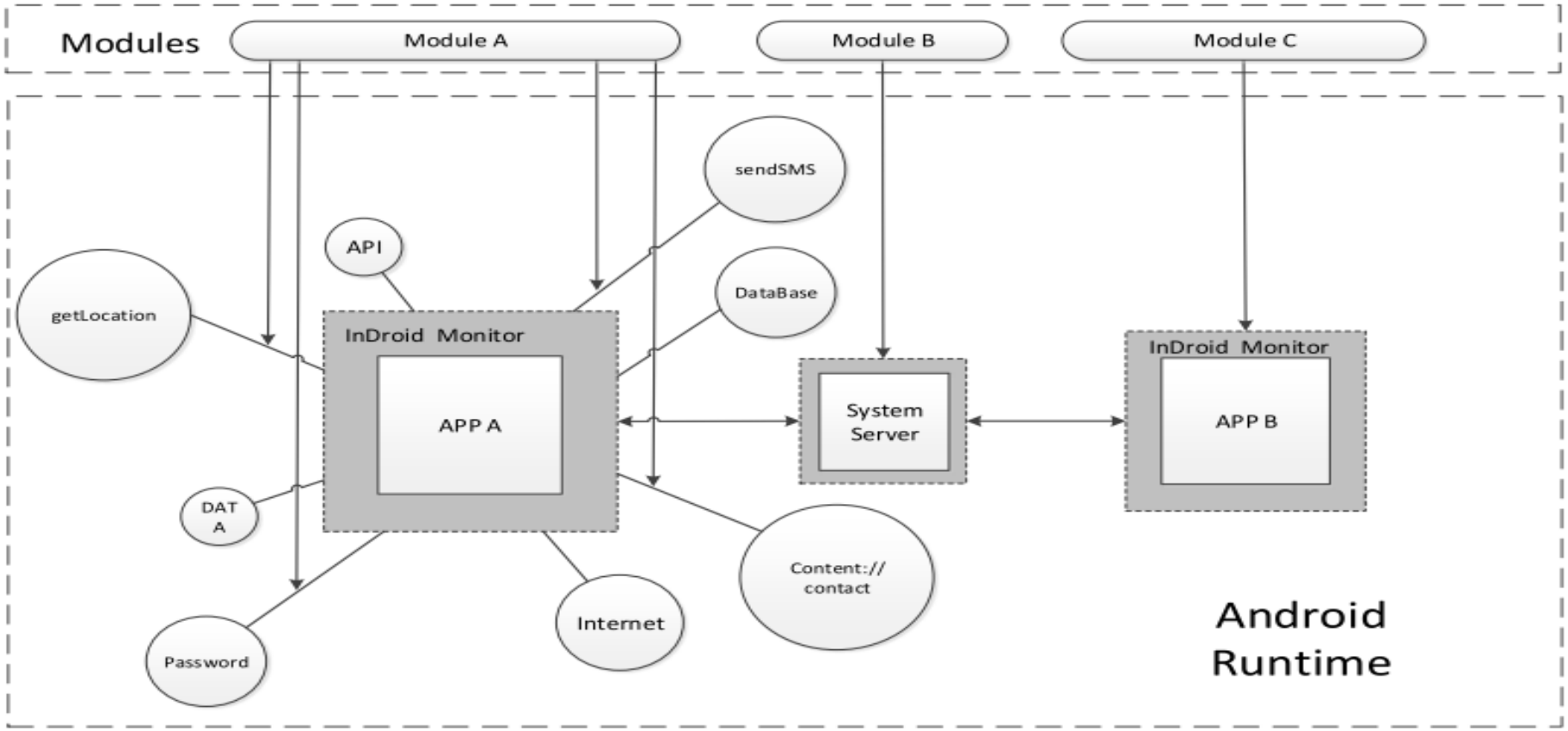
- 优势
  - —比Method hooking更深入
  - —仅需要修改libdvm.so
  - —能够快速适应Android升级（即使是ART开始流行）

InDroid：设计

- InDroid: Dalvik Instrumentation System
  - —通过修改Dalvik解释器（InterpC-portable）来实现
  - —令APP运行于Portable Interpreter下完成插桩
  - —On-demand分析，APP独立分析

- Dalvik VM Introspection
  - —通过分析Dalvik VM runtime获取上下文
  - —获取当前method
  - —获取当前所操作的Object
  - —获取当前thread

Design


Modules
Module A
Module B
Module C
sendSMS
API
DataBase
getLocation
InDroid Monitor
APP A
System Server
InDroid Monitor
APP B
DAT A
Content:// contact
Internet
Password
Android Runtime

InDroid：上下文获取

```
/* File: armv6t2/OP_MOVE.S */
/* ------------------------------ */
    .balign 64
.L_OP_MOVE: /* 0x01 */
    /* for move, move-object, long-to-int */
    /* op vA, vB */
#if defined(DIAS)
    mov r0, r4          @ r0<- program counter
    mov r1, r5          @ r1<- Frame pointer
    mov r2, r6          @ r2<- Thread pointer
    BL monitor_mov      @ Insert Probe
#endif
    mov     r1, rINST, lsr #12
    ubfx    r0, rINST, #8, #4
    FETCH_ADVANCE_INST(1)
    GET_VREG(r2, r1)
    GET_INST_OPCODE(ip)
    SET_VREG(r2, r0)
    GOTO_OPCODE(ip)
```

应用

- Trace记录
  - —可以方便地记录程序运行时的bytecode trace
  - —Function call flow

- 优点
  - —全面监控：不需要担心动态加载代码或者混淆后代码
  - —透明：应用程序感觉不到任何监控组件（性能下降除外）
  - —可扩展：Instrumentation机制允许开发复杂的分析工具

应用

- Dynamic String 监控
  - —传统的APK字符串分析只关注静态字符串
  - —InDroid更方便地记录了所有动态字符串操作
  - —通过监控StringObject以及Object中的String Class来提取字符串
  - —能获得大量字符串信息

应用

- 程序脱壳
  - —真机绕过保护措施，无法检测
  - —无需关心加壳保护方案，自动化脱壳
  - —运行时读取整体dex文件
  - —运行时获取dexfile结构体
  - —运行时获取程序真实行为

应用

- Bare Metal Comparison
  - —思路来源：BareCloud: Bare-metal Analysis-based Evasive Malware Detection @ Usenix 2014
  - —将裸机运行结果与模拟器运行结果进行记录比较

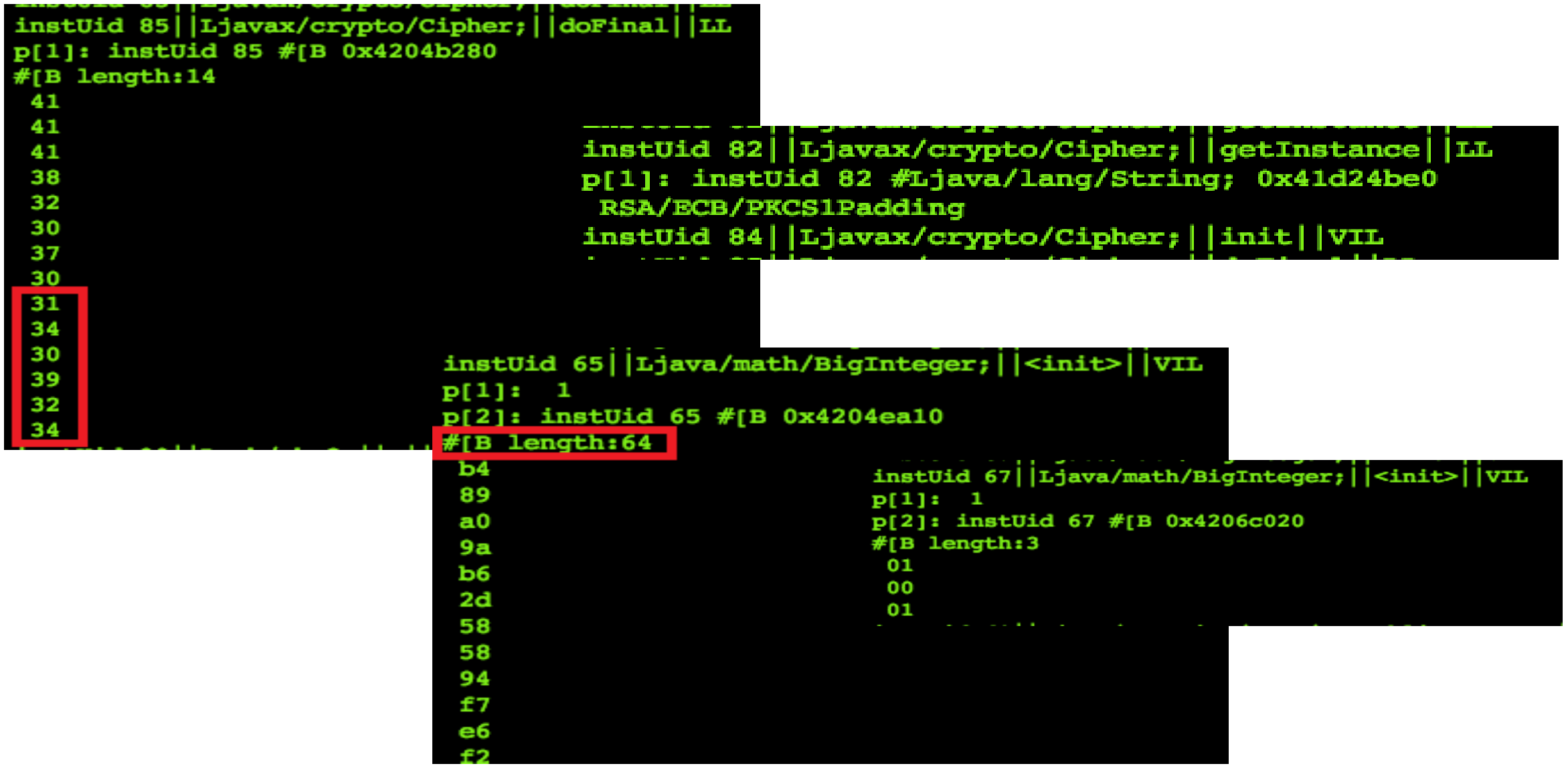
应用

- Causal Execution
  - —通过对程序运行时不同的输入导致的Trace进行分析，找到输入引起的执行改变原因

实例分析

实例分析: 某金融类APP实例分析

- 某金融类APP分析
  - 协议安全
  - 密码学误用检测


instUid 85||Ljavax/crypto/Cipher;||doFinal||LL
p[1]: instUid 85 #[B 0x4204b280
#[B length:14
41
41
41
38
32
30
37
30
31
34
30
39
32
34
instUid 82||Ljavax/crypto/Cipher;||getInstance||LL
p[1]: instUid 82 #Ljava/lang/String; 0x41d24be0
RSA/ECB/PKCS1Padding
instUid 84||Ljavax/crypto/Cipher;||init||VIL
instUid 65||Ljava/math/BigInteger;||<init>||VIL
p[1]: 1
p[2]: instUid 65 #[B 0x4204ea10
#[B length:64
b4
89
a0
9a
b6
2d
58
58
94
f7
e6
f2
instUid 67||Ljava/math/BigInteger;||<init>||VIL
p[1]: 1
p[2]: instUid 67 #[B 0x4206c020
#[B length:3
01
00
01

实例：加壳APK分析

- APK动态分析
  - 高级防护
  - 程序理解
  - 脱壳
    - 阿里
    - 梆梆
    - 360
    - …

实例：加壳APK分析

- 脱壳分析
  - 内存整体解密
    - 完整dex
  - dex结构分离
    - dexfile
  - 逐方法解密,VMP(?)
    - 行为监控

```
|[ff4a1aa4] bh.a:([BI)[B
|0000: invoke-static {}, LbKn;.a:()Z // method@08a1
|0003: move-result v3
|0004: invoke-static {v3}, LbKn;.b:(I)V // method@08a2
|0007: add-int/lit8 v0, v5, #int 1 // #01
|0009: invoke-static {v4, v5}, Lcd;.a:([BI)[B // method@0b23
|000c: move-result-object v1
|000d: add-int/lit8 v2, v0, #int 1 // #01
|000f: invoke-static {v1, v0}, LcC;.a:([BI)[B // method@0a30
|0012: move-result-object v0
|0013: add-int/lit8 v1, v2, #int -1 // #ff
|0015: invoke-static {v0, v2}, Lp;.a:([BI)[B // method@0e8d
|0018: move-result-object v0
|0019: invoke-static {v0, v1}, Lx;.a:([BI)[B // method@0ede
|001c: move-result-object v0
|001d: add-int/lit8 v2, v1, #int -1 // #ff
|001f: invoke-static {v0, v1}, Lali$a;.M$d:([BI)[B // method@03d3
|0022: move-result-object v0
|0023: add-int/lit8 v1, v2, #int 1 // #01
|0025: invoke-static {v0, v2}, LaS;.a:([BI)[B // method@022e
|0028: move-result-object v0
|0029: invoke-static {v0, v1}, Lx;.a:([BI)[B // method@0ede
|002c: move-result-object v0
|002d: add-int/lit8 v2, v1, #int 1 // #01
|002f: invoke-static {v0, v1}, Lali$a;.M$z:([BI)[B // method@0440
|0032: move-result-object v0
```

Thanks !