基于真实Android环境下的APP程序分析与安全检测

China Internet Security Conference 2014

2014中国互联网安全大会

提纲

1. APP分析的现状与工具
2. 基于模拟器分析的不足之处
3. APP高级程序分析需求
4. 基于真实环境的APP分析
5. 实例分析

提纲

1. **APP分析的现状与工具**
2. 基于模拟器分析的不足之处
3. APP高级程序分析需求
4. 基于真实环境的APP分析
5. 实例分析

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

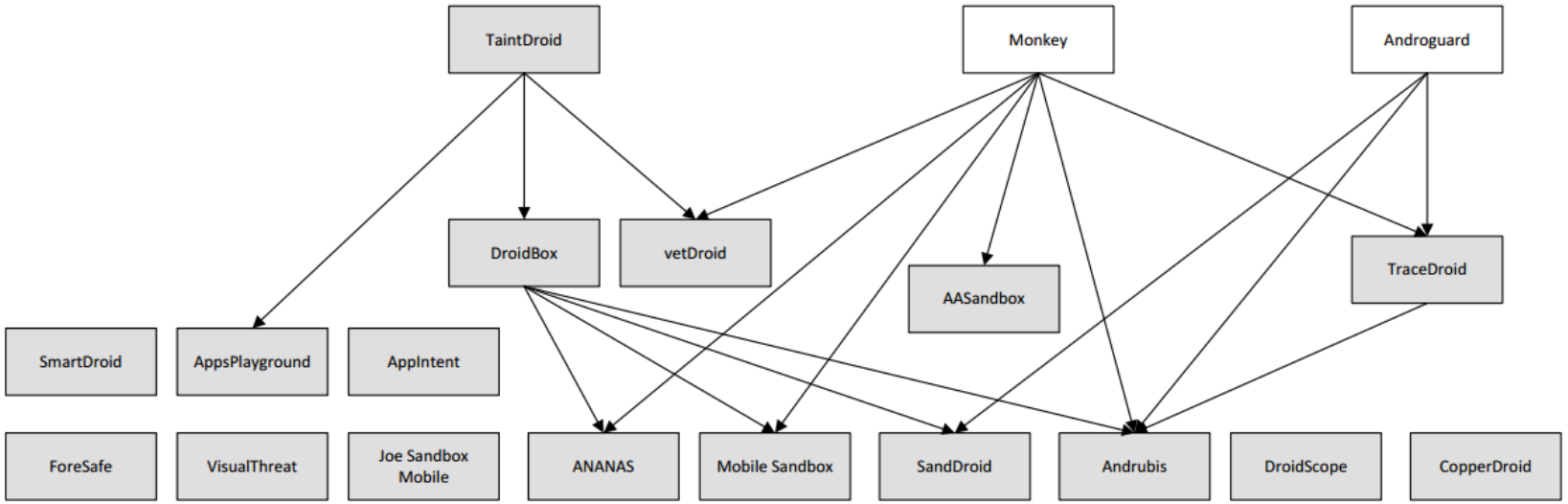
- 动态分析
  - 对抗代码加壳、混淆等
  - 监控各类事件

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



- *图片来源：https://www.sba-research.org/wp-content/uploads/publications/mostAndroid.pdf*

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

提纲

1. APP分析的现状与工具
2. **基于模拟器分析的不足之处**
3. APP高级程序分析需求
4. 基于真实环境的APP分析
5. 实例分析

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

提纲

1. APP分析的现状与工具
2. 基于模拟器分析的不足之处

3. APP高级程序分析需求

4. 基于真实环境的APP分析
5. 实例分析

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

提纲

1. APP分析的现状与工具
2. 基于模拟器分析的不足之处
3. APP高级程序分析需求

4. 基于真实环境的APP分析

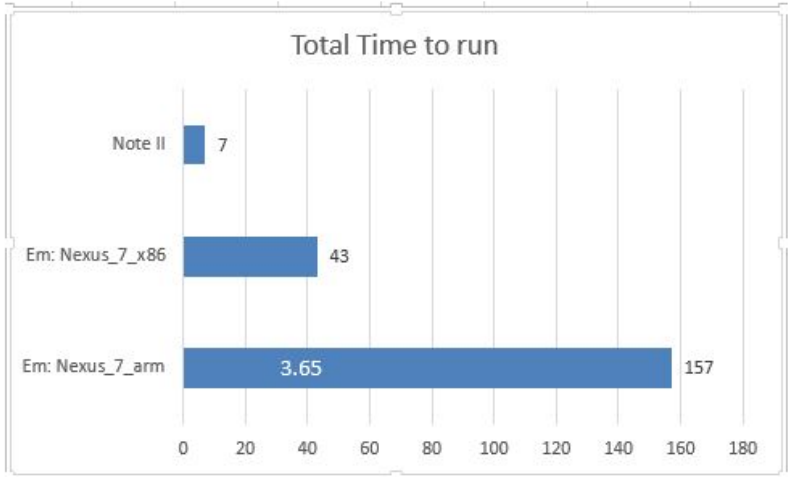
5. 实例分析

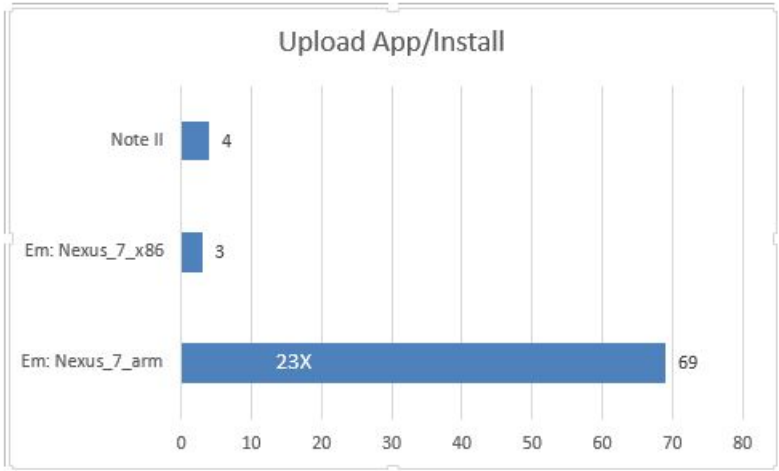
真实环境能带来哪些好处

- 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

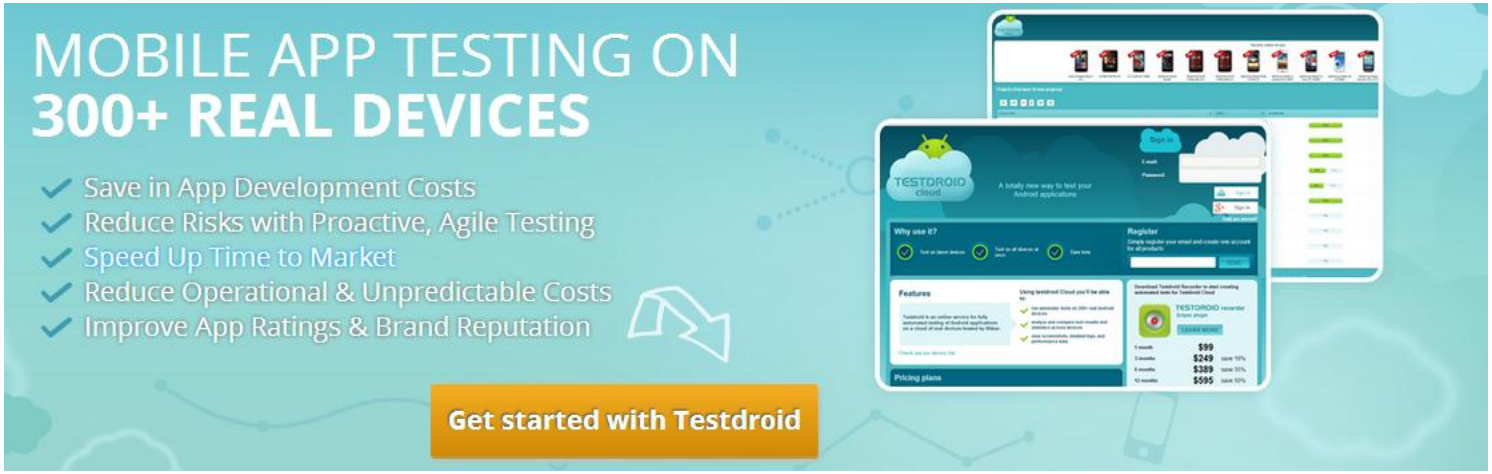




真实环境能带来哪些好处

- 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



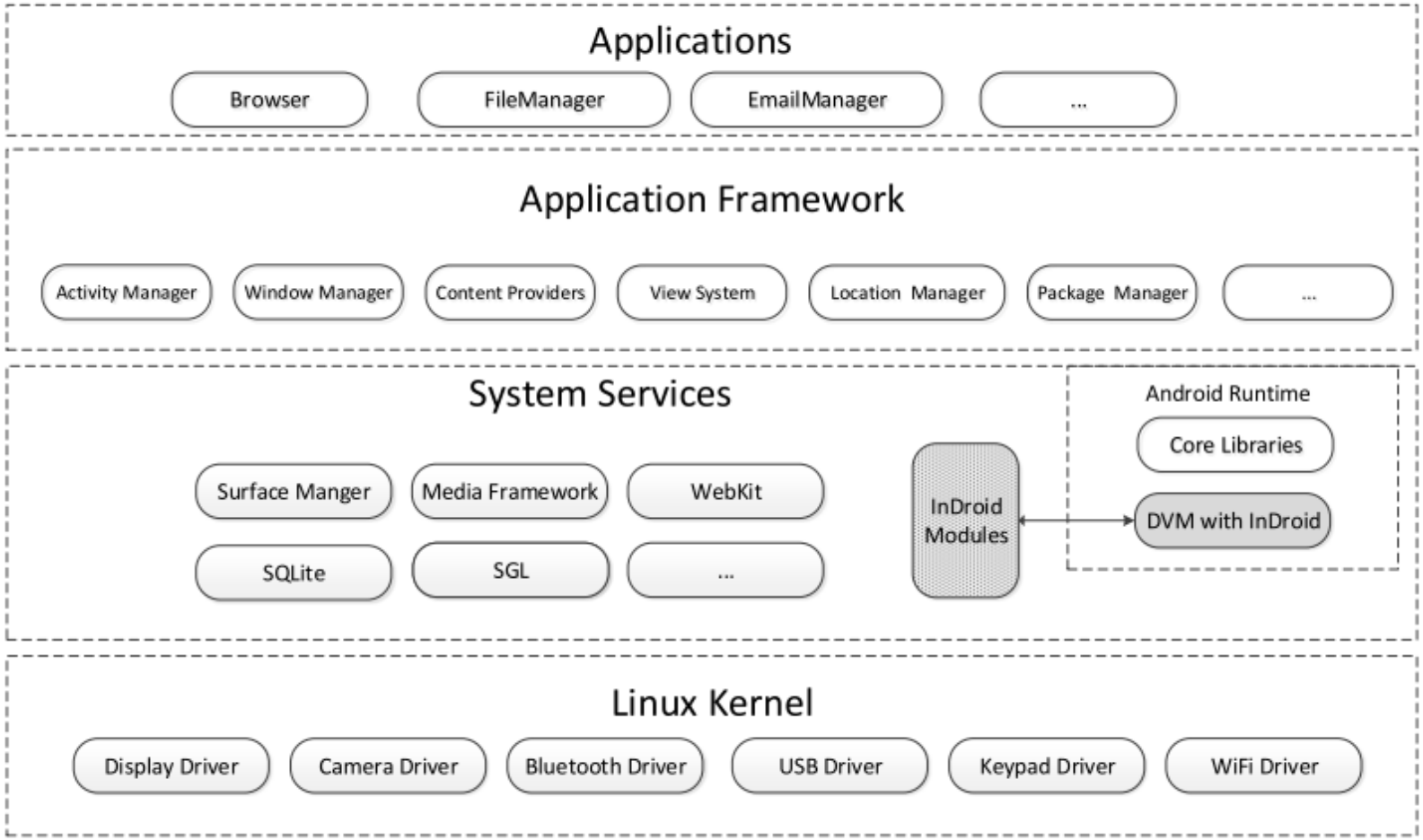
真实环境能带来哪些好处

- 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



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

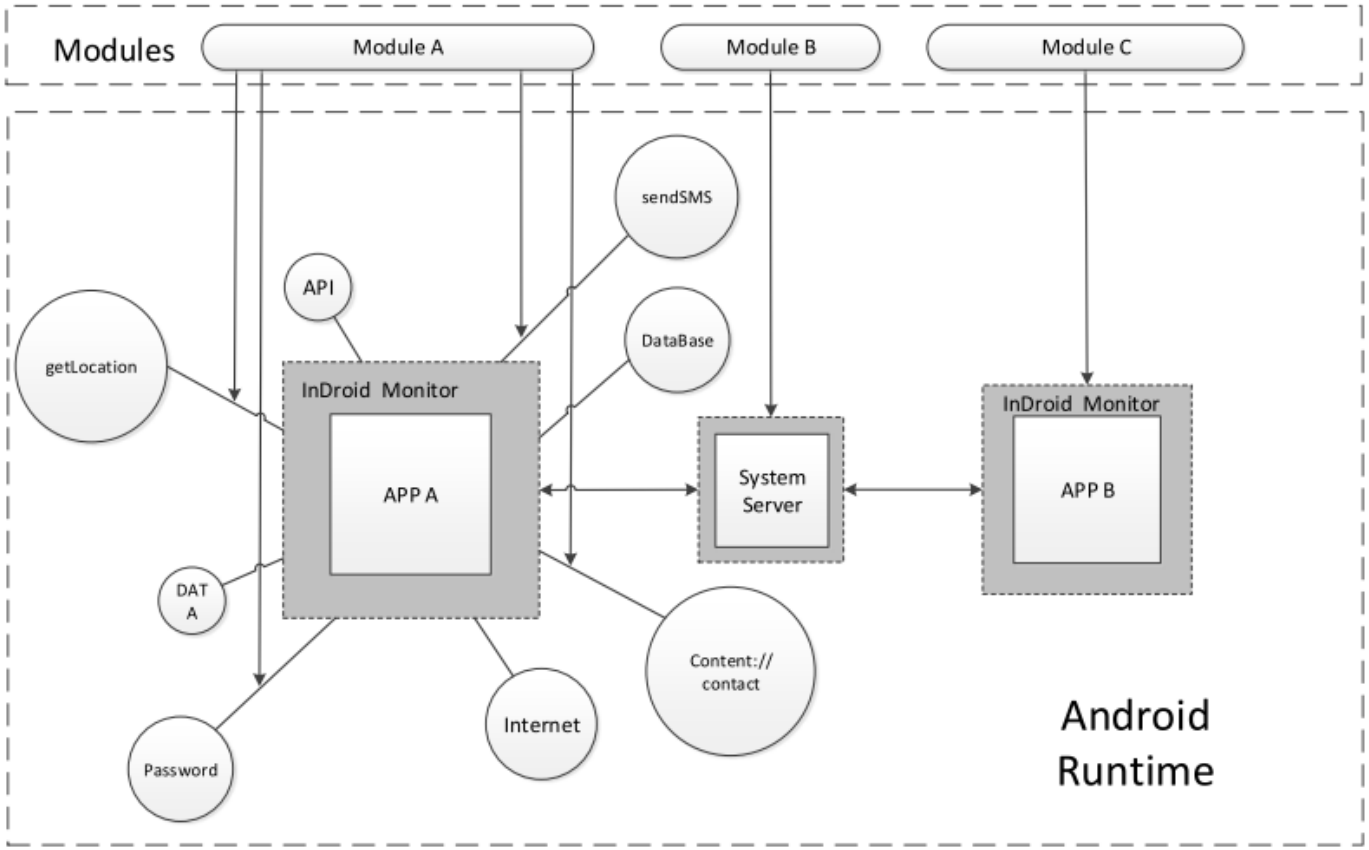
- 改造Dalvik VM
  - 思路，为DVM提供类似JVM Tool Interface的接口
  - 支持bytecode级别指令级插桩监控
- 优势
  - 比Method hooking更深入
  - 仅需要修改libdvm.so
  - 能够快速适应Android升级（即使是ART开始流行）

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

– 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

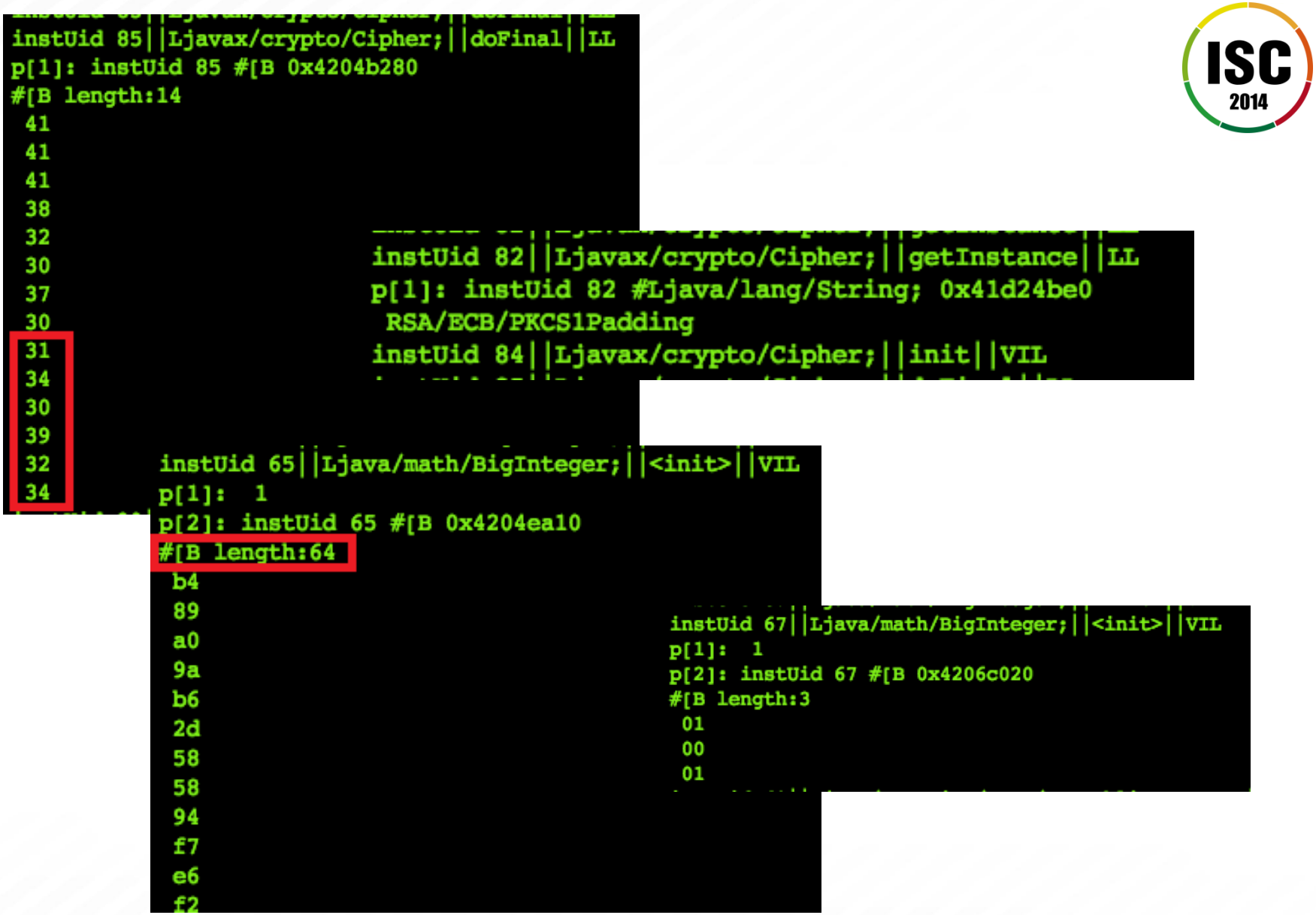
提纲

1. APP分析的现状与工具
2. 基于模拟器分析的不足之处
3. APP高级程序分析需求
4. 基于真实环境的APP分析

5. 实例分析

实例分析: 银行类APP实例分析

- 银行类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: AliCTF APK分析

- APK动态分析
  - 高级防护
  - 加壳保护
  - 程序理解

Thanks!