Android应用层虚拟化恶意代码的分析与检测

汇报人：傅 建 明

单位：武汉大学国家网络安全学院
空天信息安全与可信计算教育部重点实验室

汇报提纲

SYNOPSIS

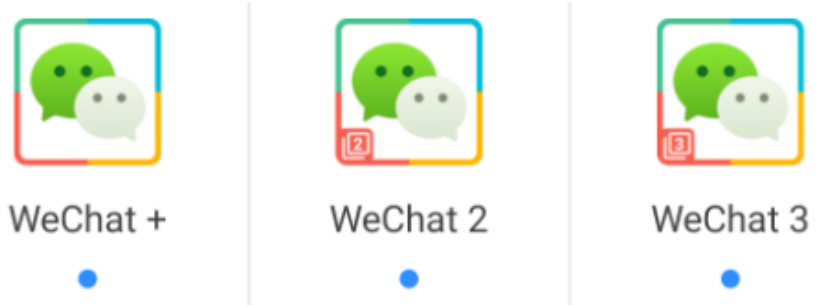
01 应用层虚拟化框架

02 VA框架的分析与识别

03 VA病毒的分析与检测

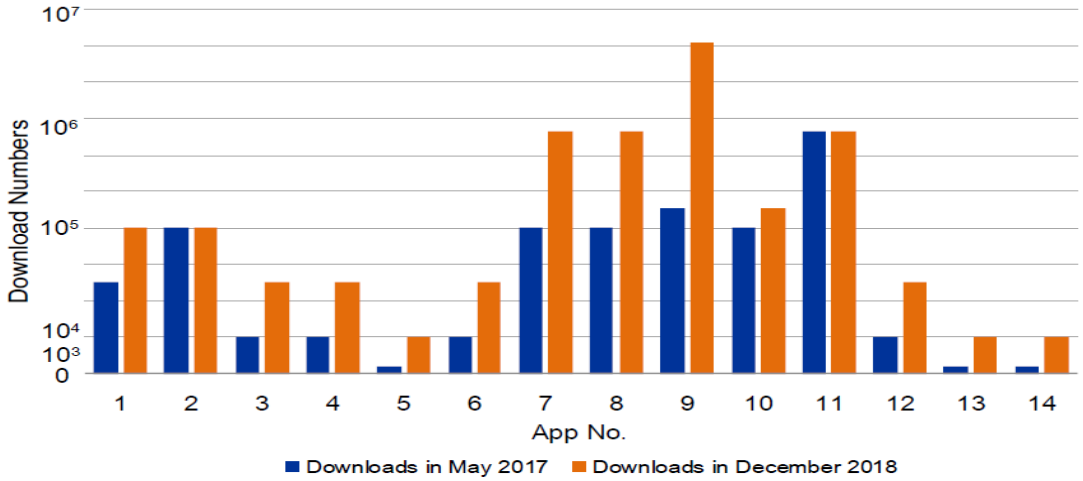
1、应用层虚拟化框架

- 多个社交账号并发使用
  - 多个设备
  - 重打包
  - 多开应用（应用层虚拟化-VA）
- VA框架
  - 突破DEX的方法数限制（ 65535 ）
  - 热修复
  - 模块化开发



VirtualApp
DroidPlugin

1、应用层虚拟化框架



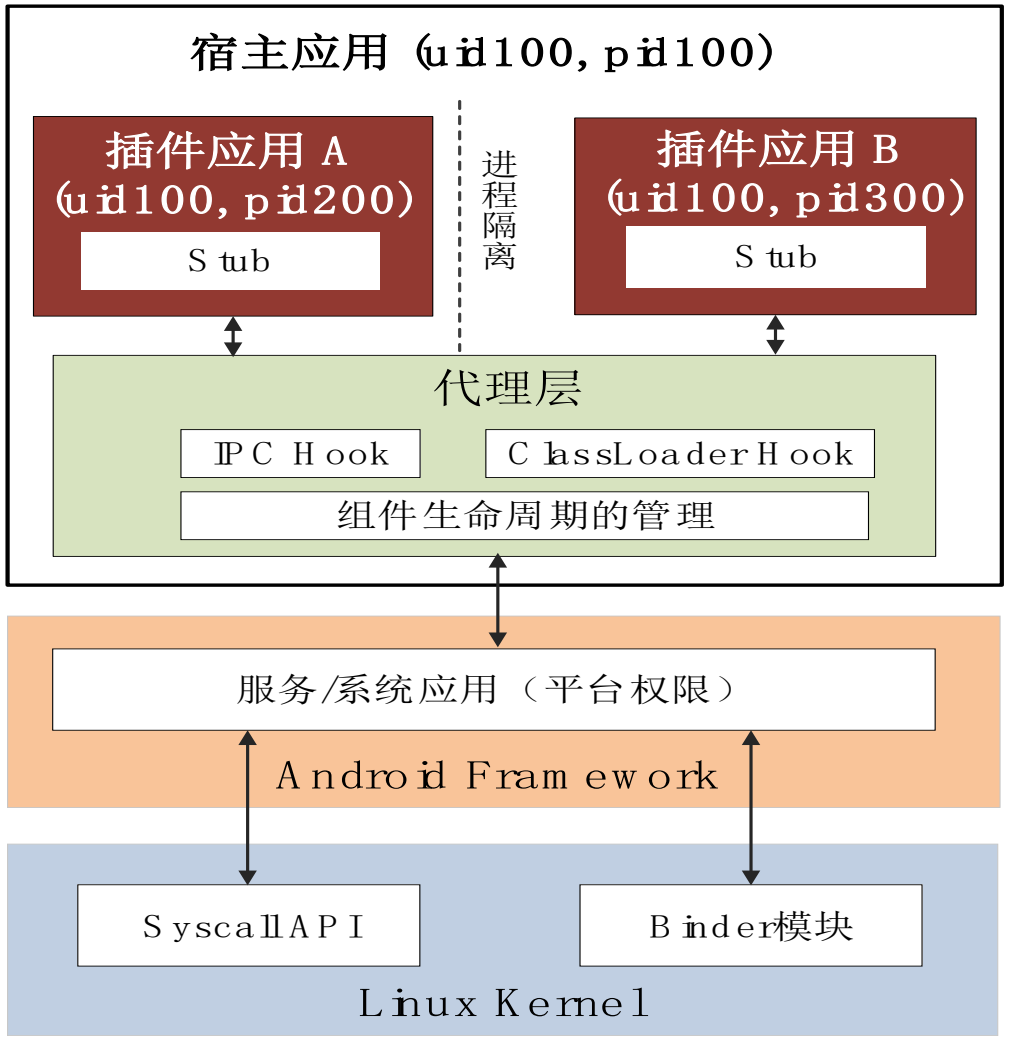
- VA框架的流行度

应用名称	平行空间	双开空间	多开空间	双开助手	应用分身	Dual Apps	多开大师	2Accounts
下载量	1亿+	1亿+	1000万+	1000万+	1000万+	1000万+	500万+	500万+
提供方	LBE Tech	DUALSPACE	Winterfell Applab	MA Team	Arty Product	Hide Apps	River Stone Tech	2Accounts
版本号	4.0.9078	4.0.2	1.5.40.0329	3.3.9	1.1.4.3	2.9.2_703d758f7	2.43.06.0331	3.3.5
更新时间	2021-03-24	2021-03-24	2021-03-29	2021-03-24	2021-01-27	2021-03-11	2021-03-31	2021-03-24

1、应用层虚拟化框架

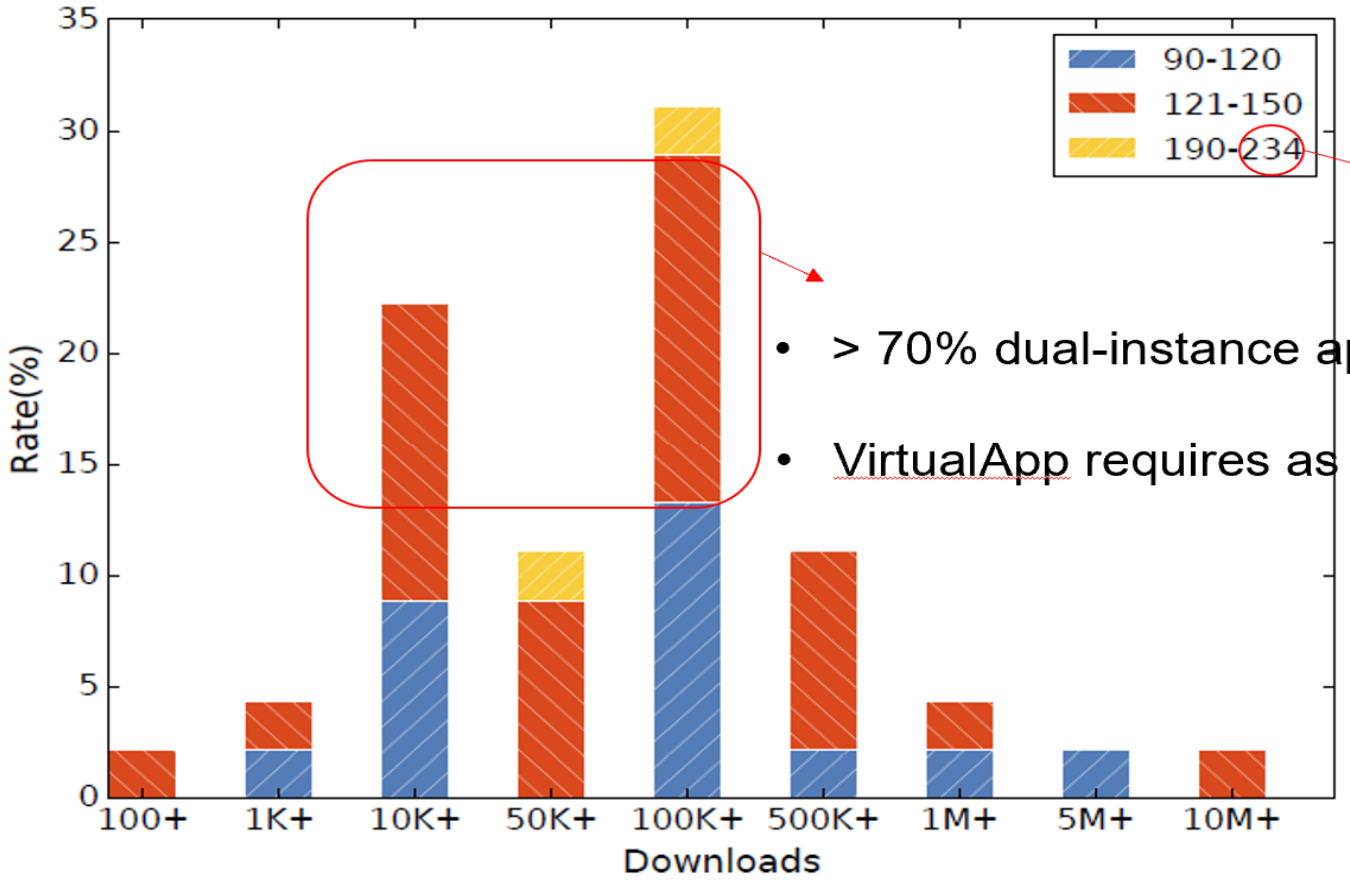
- VA框架
  - Host app
  - Guest/plugin app

- VA机理
  - 权限复用
  - UID共享
  - I/O重定向
  - 服务代理
  - 组件占坑



1、应用层虚拟化框架

- VA权限申请



- 129 permissions on average
- Max number: 234
- > 70% dual-instance apps have > 120 permissions
- VirtualApp requires as much as 186 permissions

汇报提纲

SYNOPSIS

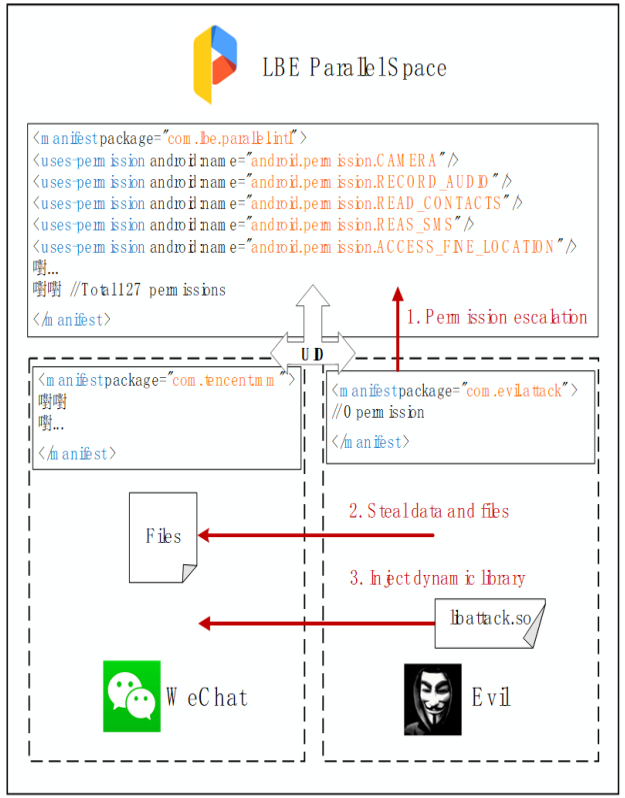
01 应用层虚拟化框架

02 VA框架的分析与识别

03 VA病毒的分析与检测

2、VA框架的分析与识别

- VA框架的应用
  - 双开软件
  - 嵌入式服务
  - 游戏加载
- VA框架的安全分析
  - 组件劫持
  - 权限滥用
  - 信息泄露



Host app and guest apps share a same UID.

◆ **Permission Escalation**

Use unapplied permissions

◆ **Privacy Leakage**

Steal sensitive files and data

◆ **Component Hijacking**

Intercept functions

- [MobiSys '19] "Jekyll and Hyde" is Risky: Shared-Everything Threat Mitigation in Dual-Instance Apps.
- [Sigmetrics ' 19] App in the Middle: Demystify Application Virtualization in Android and its Security Threats.
- [BlackHat2017] Anti-Plugin: Don't Let Your App Play as an Android Plugin

2、VA框架的分析与识别

- VA框架的安全分析
  - 组件劫持
  - 权限滥用
  - 信息泄露


amazon.png


shopee.png


淘宝.png


amongus.png


绝地求生.png


王者荣耀.png


instagram.png


twitter.png


whatsapp.png

应用名称	平行空间	双开空间	多开空间	双开助手	应用分身	Dual Apps	多开大师	2Accounts
权限滥用	Y	Y	Y	Y	Y	Y	Y	Y
信息泄露	Y	Y	Y	Y	Y	Y	Y	Y

2、VA框架的分析与识别

[MobiSys '19]

- VA框架的动态识别
  - app运行在VA会存在安全风险

Category	Feature
Path	1. Host app's APK path in guest app process
	2. APK source code and dynamic-link library path
UID	3. Multiple processes with the same UID
	4. Check undeclared permissions
Code Injection&	5. Stack tracking of exception
Hooking	6. Suspicious library to provide native hooking

DiPrint

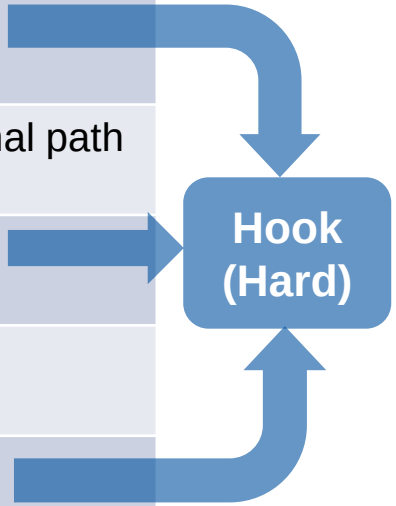
Open-sourced DiPrint (https://github.com/whucs303/DiPrint)

2、VA框架的分析与识别

[MobiSys '19]

- 特征的鲁棒性

Feature	Bypassing ways
1. Host app's APK path in guest app process	
2. APK source code and dynamic-link library path	Read files from the original path **(lack stability)**
3. Multiple processes with the same UID	
4. Check undeclared permissions	**Cannot be bypassed !**
5. Stack tracking of exception	
6. Suspicious library to provide native hooking	Use JAVA hooking mechanism instead of native hooking



2、VA框架的分析与识别

[MobiSys '19]

- 实验结果

Name	Number	1. Exist. host APK	2. APK/Lib path	3. Processes	4. Permissions	5. Stack tracking	6. Hooking lib	Time[1] (ms)	Virtualization Engine
Real Android Systems[2]	109	0	0	0	0	0	0	0.22	None
Commercial Dual-Instance Apps									
Category 1	15	15	15	15	15	15	13	0.41	VirtualApp
Category 2	5	5	3	5	5	5	3	0.28	DroidPlugin
Category 3	2	2	2	2	2	2	2	0.52	MultiDroid
Category 4	20	20	14	20	20	20	13	0.33	Custom-made Engines[3]
Android Malware Samples									
Category 5	294	294	294	294	294	294	147	0.29	VirtualApp
Category 6	63	63	63	63	63	63	42	0.37	DroidPlugin
Category 7	24	24	24	24	24	24	16	0.43	Custom-made Engines
Non-Virtualization-Based Malware	1,862	0	0	0	0	0	0	0.21	None
Representative Examples									
Boxify[5]		Y	Y	N	Y	Y	Y	N/A[4]	
NJAS[9]		Y	Y	N	N	Y	Y	N/A[4]	
com.in.parallel.accounts (GooglePlay)		Y	Y	Y	Y	Y	Y	0.32	VirtualApp
com.ludashi.dualspace (GooglePlay)		Y	N	Y	Y	Y	Y	0.29	VirtualApp
com.qihoo.magic (360)		Y	Y	Y	Y	Y	Y	0.14	DroidPlugin
com.lbe.parallel.intl (GooglePlay)		Y	Y	Y	Y	Y	Y	0.26	MultiDroid
com.youlong.multiaccount (GooglePlay)		Y	N	Y	Y	Y	Y	0.36	Custom-made Engine
com.excelliance.multiaccount (GooglePlay)		Y	Y	Y	Y	Y	Y	0.33	Custom-made Engine
Malware 1: com.twittre.android		Y	Y	Y	Y	Y	Y	0.23	VirtualApp
Malware 2: PluginPhantom		Y	Y	Y	Y	Y	Y	0.41	DroidPlugin
Active Attacker[5]		N	N	N	Y	Y	N	0.81	VirtualApp

汇报提纲

SYNOPSIS

01 应用层虚拟化框架

02 VA框架的分析与识别

03 **VA病毒的分析与检测**

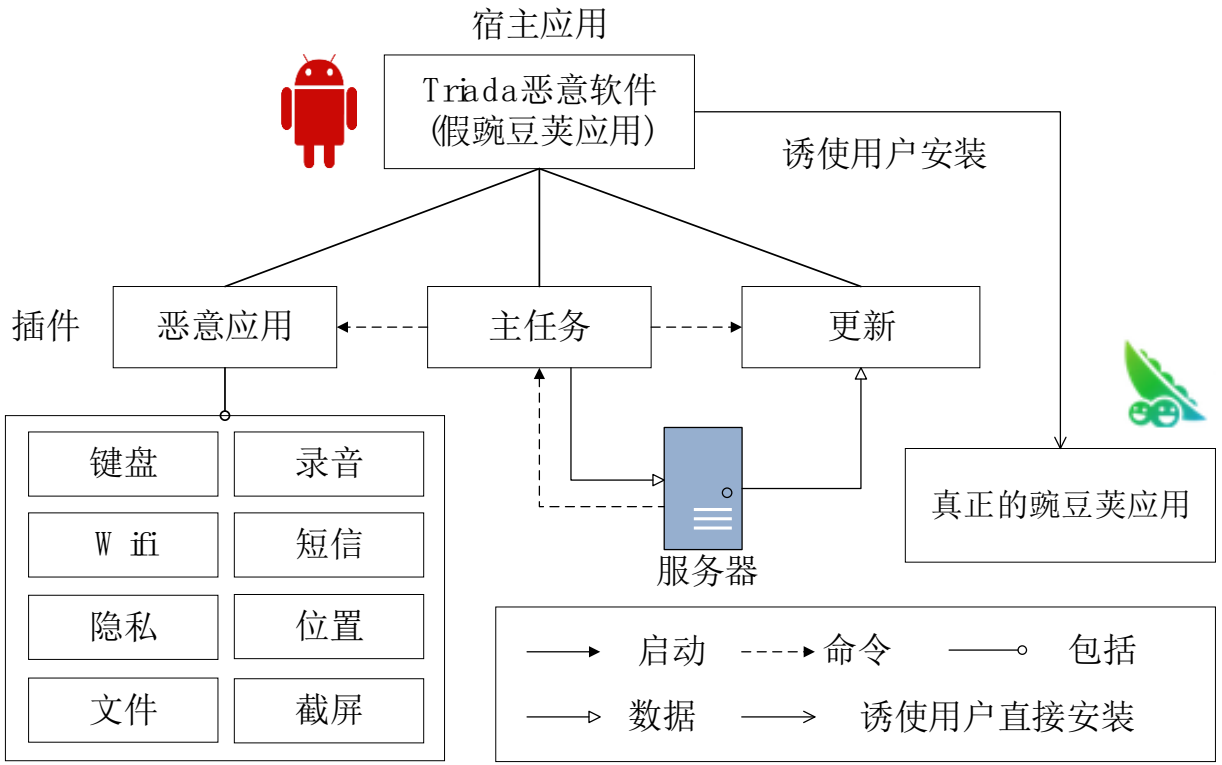
3、VA病毒的分析与检测

- VA病毒-**PluginPhantom/HummingBad**
  - Bypass static analysis
  - Rely on DroidPlugin to install malware from “Assets” directory or from Internet

- New **adware** abuses dual-instance apps to automatically launch different advertisement apps without user interactions.

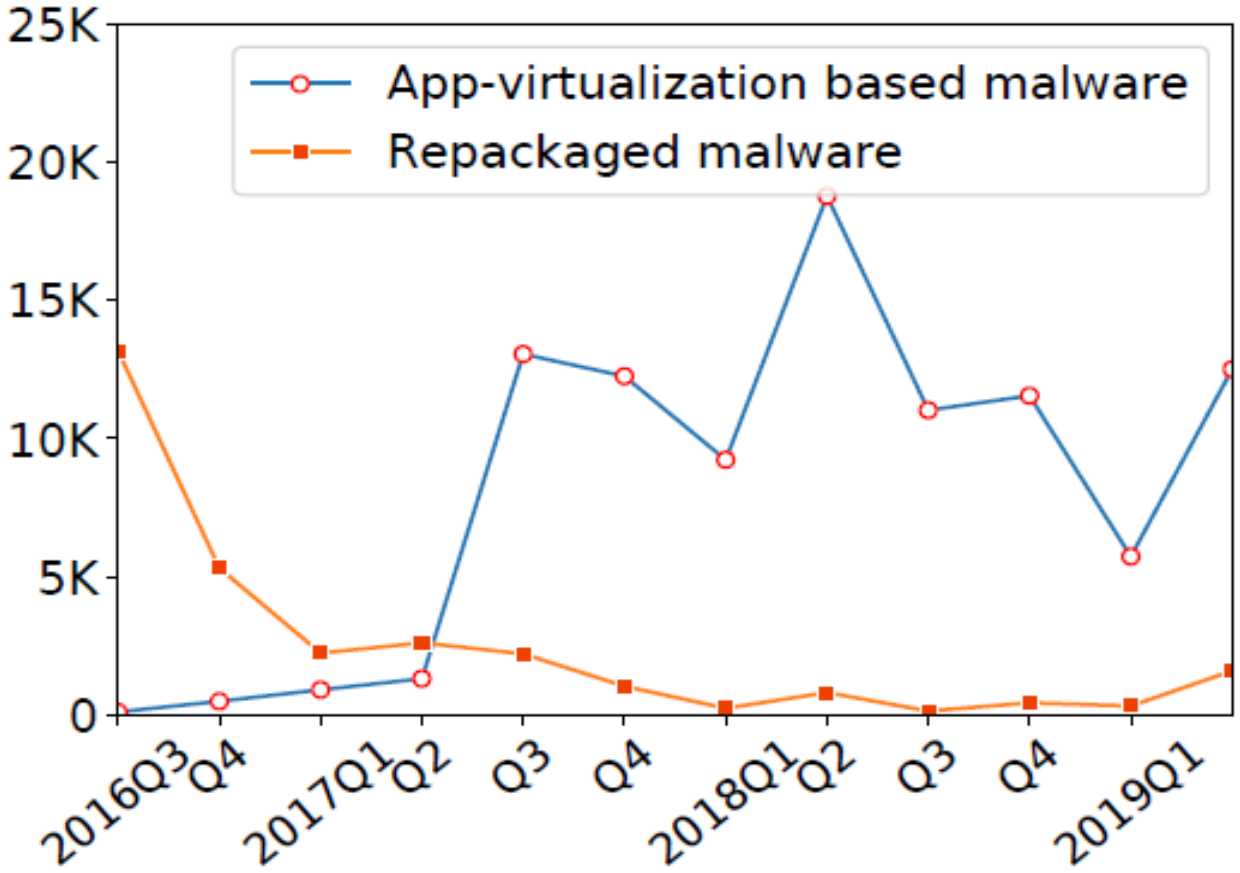
3、VA病毒的分析与检测

- VA病毒-Triada病毒
  - 从重打包 —> 虚拟化加载
  - 动态分发，每个插件功能独立
  - 伪装性强



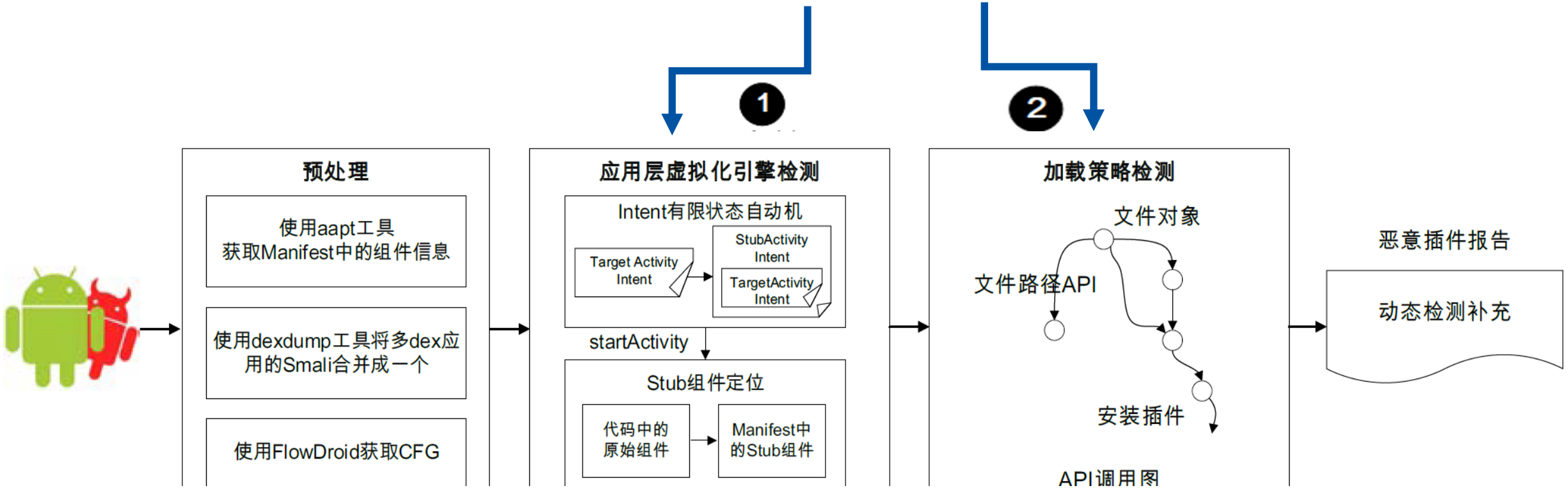
3、VA病毒的分析与检测

- VA病毒
  - 恶意代码加载
  - 推送广告
  - 色情传播
- VA病毒检测
  - 动态
  - **静态**



3、VA病毒的分析与检测

- 两步走检测思路

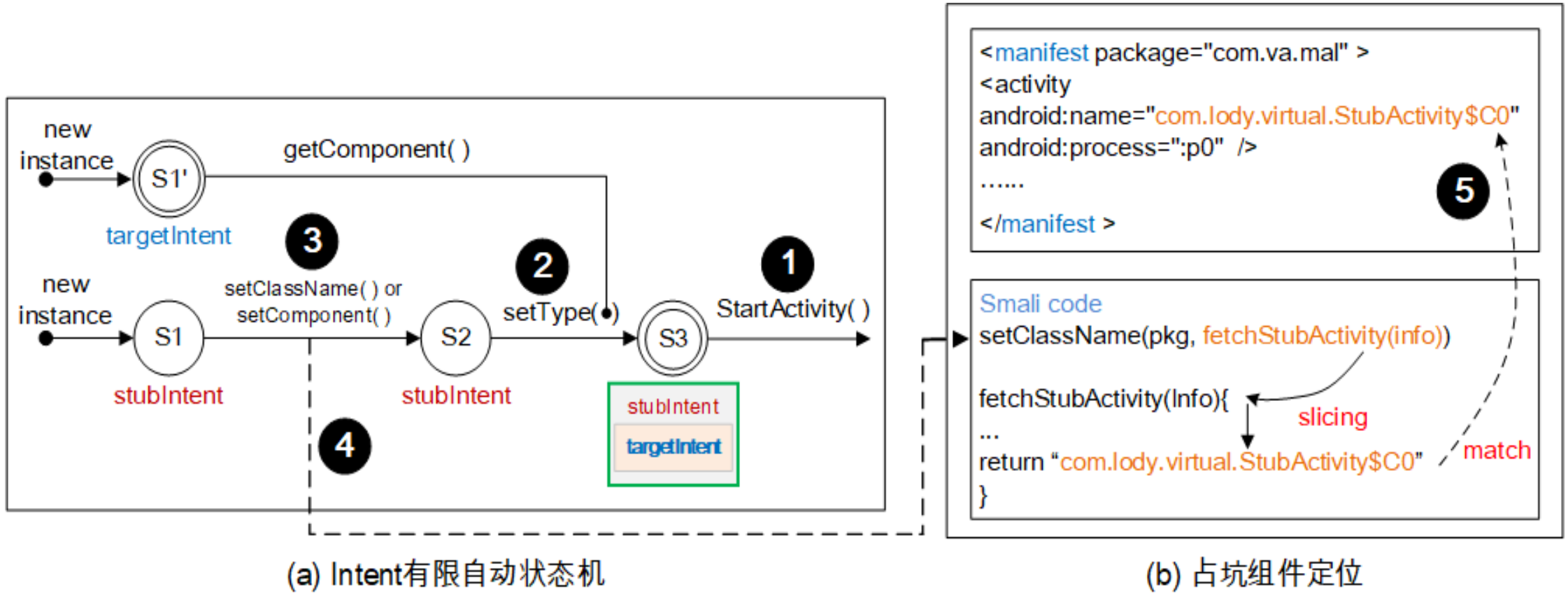


[CCS'20] VAHunt: Warding Off New Repackaged Android Malware in App-Virtualization's Clothing

3、VA病毒的分析与检测

- ~~权限复用~~
- ~~UID共享~~
- ~~I/O重定向~~
- ~~服务代理~~
- 组件占坑

● VA引擎检测



(a) Intent有限自动状态机　　(b) 占坑组件定位

3、VA病毒的分析与检测

● VA引擎检测

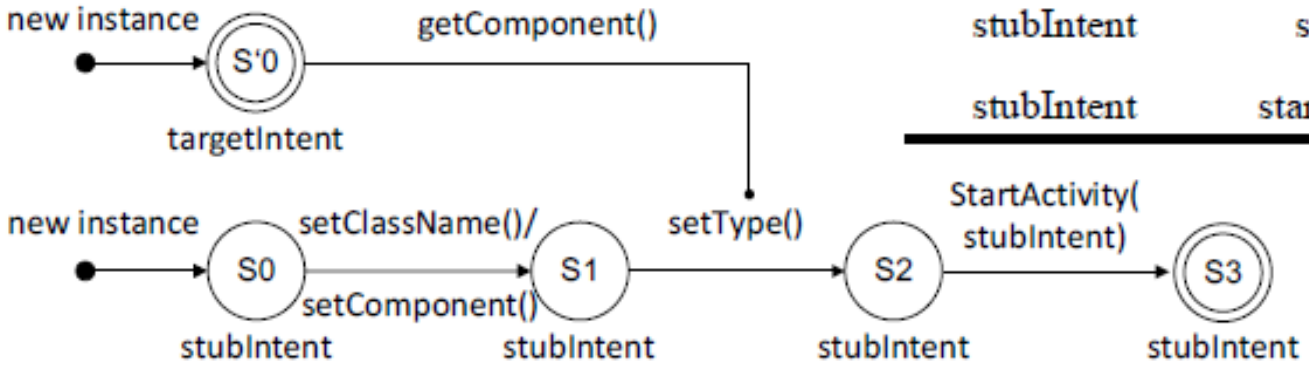
```
private Intent startActivityProcess(Intent targetIntent)
{
    intent = new Intent(targetIntent);
    Intent stubIntent = new Intent();
    stubIntent.setClassName(PackageName, StubActivity);
    ComponentName component = intent.getComponent();
    stubIntent.setType(component);
    startActivity(stubIntent);
}
```

□ Intent替换

➢ 为每个Intent创建自动机

1.代码 -> Intent查找表

Intent	操作	参数	返回值
targetIntent	getComponent	null	component
stubIntent	setClassName	getHostPkg(), fetchStubActivity()	null
stubIntent	setType	flattenToString(component)	null
stubIntent	startActivity	null	null



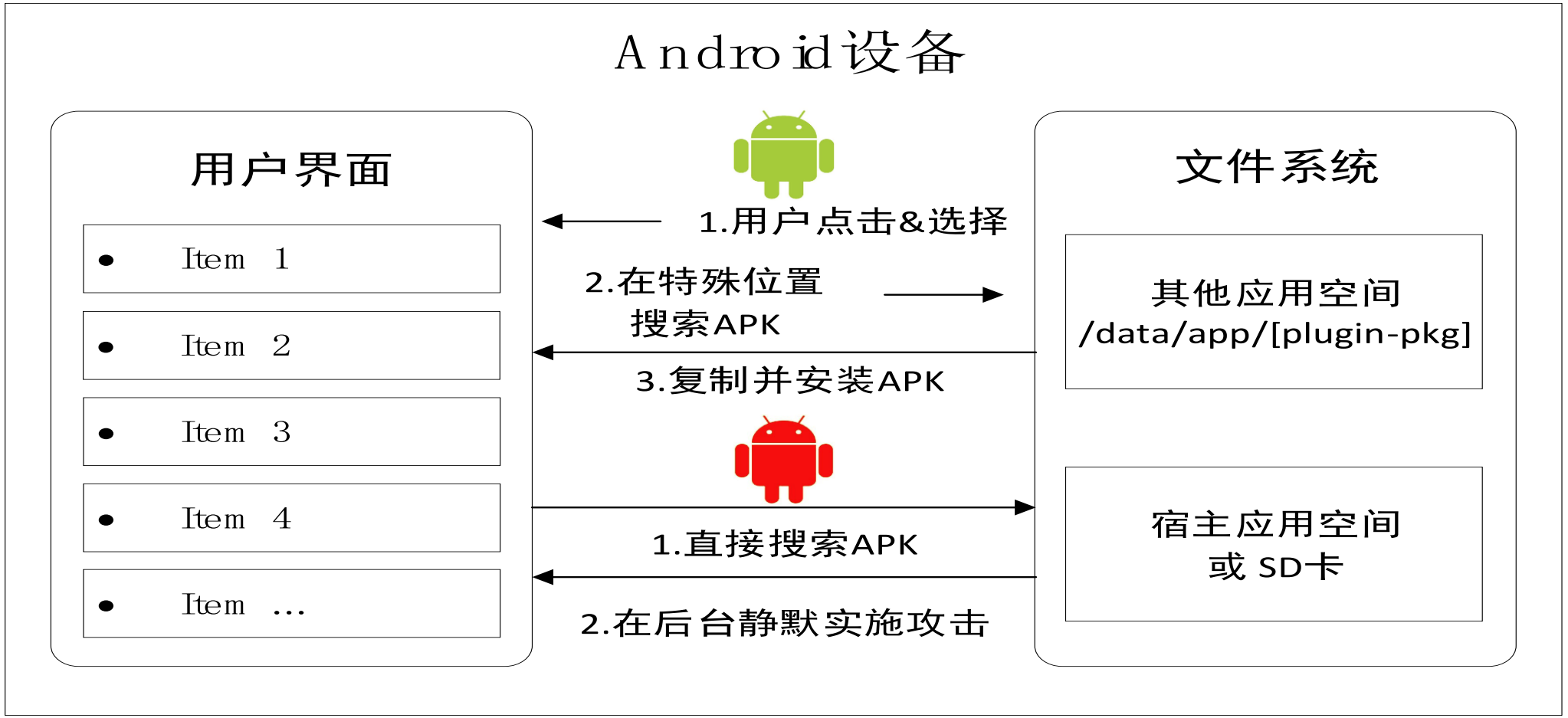
2.查找表 -> Intent自动机

3、VA病毒的分析与检测

● 插件加载策略

□ 用户交互

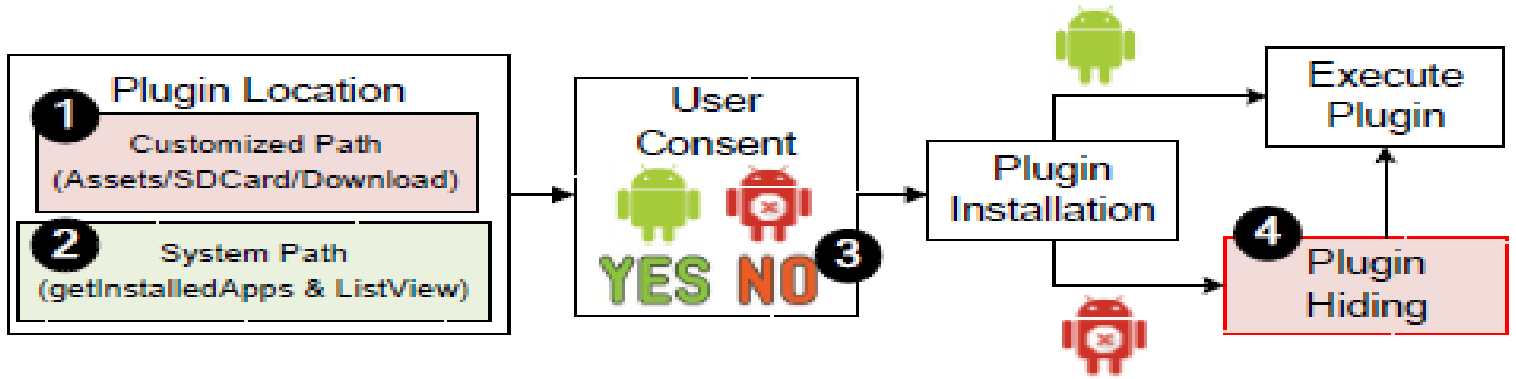
□ 插件来源



3、VA病毒的分析与检测

● 插件加载策略

特征	分析初始点	分析结束点
1.安装路径	敏感路径 API	installApp/installPackage
2.安装列表	getInstalledPackages() exec("pm list packages")	onCreateView()/onViewCreated()/ inflate()/onCreateViewHolder()/setContentView()
3.直接安装	installApp/ installPackage	onClick()/setOnItemClickListener() Manifest ("android.intent.category.LAUNCHER")
4.图标隐藏	任何点	Window.addFlag(FLAG_NOT_TOUCH_MODAL) Window.setFlag(FLAG_NOT_TOUCH_MODAL) Window.requestFeature(FLAG_NOT_TOUCH_MODAL) setComponentEnabled(comp, 2, 1)



◆ silentInstall:

- (1 OR 2) AND 3

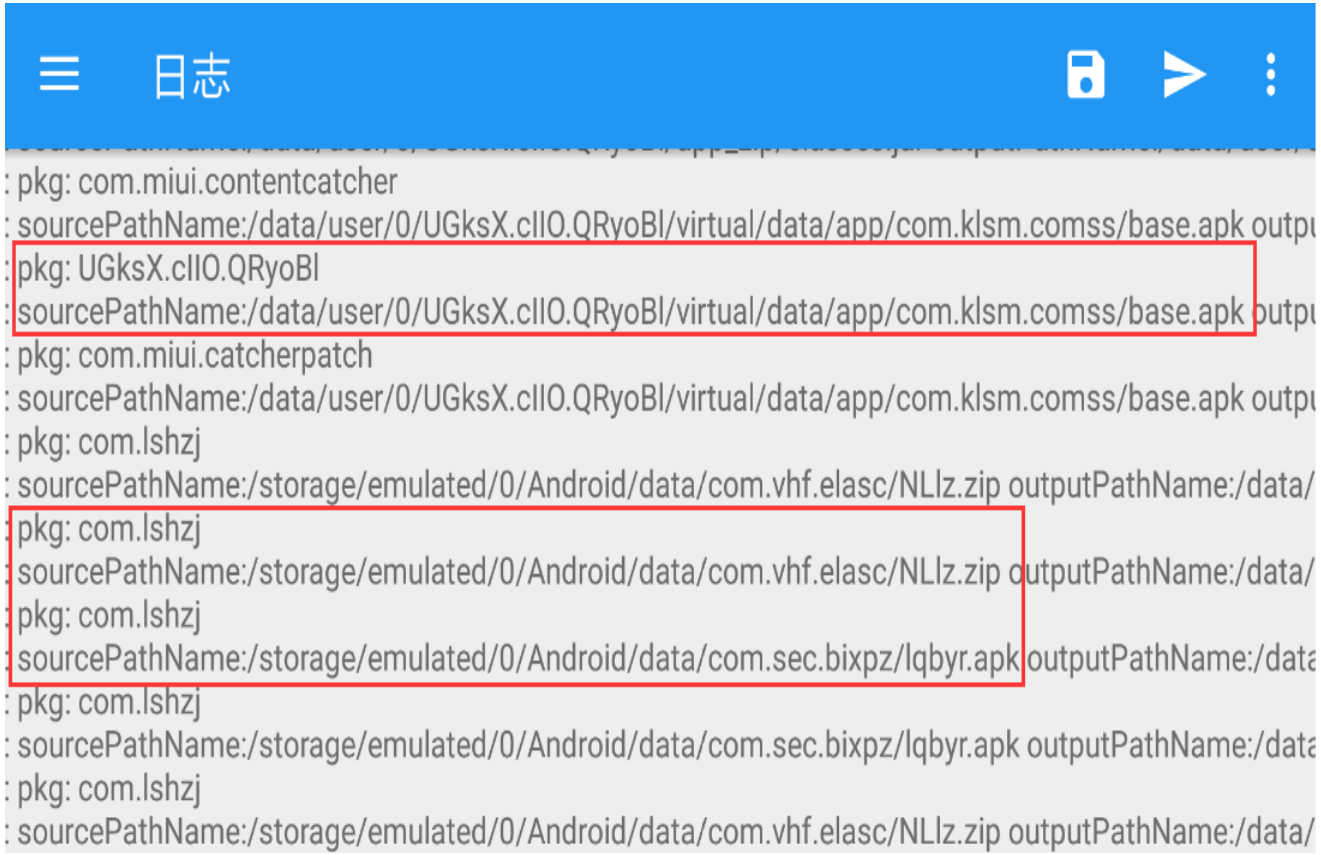
◆ silentInstall+:

- silentInstall AND 4

3、VA病毒的分析与检测

● 动态检测

- Xposed - Hook函数
  - Intent操作函数：setClassName()、getComponent()和setType()、setData()、setDataAndType()、putExtra()和putExtras()
  - 动态加载函数：DexFile Class的loadDex(String sourcePathName, String outputPathName, int flags)



3、VA病毒的分析与检测

- 实验结果

Virtu. Engine	Category	Number	VAHunt1[1]	VAHunt2[2]	Certificate [16]
VirtualApp	Benign	18, 508	100%	100%	11.7%
	Malware	62, 717		99.3%	0.3%
DroidPlugin	Benign	44, 498	100%	100%	17.1%
	Malware	3, 014		98.7%	0.5%
Custom-made	Benign	8, 261	100%	100%	13.7%
	Malware	2, 360		99.2%	0.3%
Dual-Instance	Benign	147	100%	100%	0
No app-virtu.	Benign	1, 638	100%	-[3]	-[3]
	Malware	3, 212			
Overall	Benign	73, 052	100%	100%	14.9%
	Malware	71, 303		99.3%	0.3%
Avg. Time			6.9s	12.1s	0.6s

- 漏报
  - ✓ 加壳（百度加密、梆梆加密等） 496个样本
  - ✓ 二次加载 31个样本

3、VA病毒的分析与检测

- 实验结果

Virtualization Engine	Category	customizedPath	systemPath	silentInstall	silentInstall+[1]
Dual-Instance	Benign	0	100%	0	0
VirtualApp	Benign	93.5%	6.5%	0	0
	Malware	99.6%	0.4%	99.3%	7.6%
DroidPlugin	Benign	90.2%	9.8%	0	0
	Malware	99.9%	0.1%	98.7%	11.7%
Custom-made	Benign	86.3%	13.7%	0	0
	Malware	99.7%	0.3%	99.2%	9.8%

[1] silentInstall+ means host app loads plugins silently and hides app.

3、VA病毒的分析与检测

- 实验结果

Category	No Plugin	Encrypted Plugin	Plain-text Plugin[1]
benign	76.8%	3.7%	19.5%
malicious	10.5%	89.2%	0.3%

[1]Plain-text plugins are APK files that are not encrypted.

- 大多数良性VA在本地没有插件
- 大多数恶意VA在本地有插件，且插件是加密的

3、VA病毒的分析与检测

- 鲁棒性

对抗类型	混淆方法	对抗的鲁棒性	对检测是否有影响
对抗基于 API 的检测	1.插入 dummy API	普通	无
	2.插入 dummy bytecode	普通	无
	3.包名加密	强	无
	4.将组件代码分块	无	无
对抗基于数据流的检测	5.隐式数据流	无	无
	6.JAVA 反射	强	有
对抗基于证书的检测	7.字符串加密	强	有
	8.变量名加密	普通	无
	3.包名加密	强	无
	9． Class/Method 名加密	强	无

总结

- Android的应用层虚拟化框架（VA）为双开应用带来了便利
  - 大号、小号可以同时使用
- 良性双开VA因权限隔离和数据隔离不足给插件app带来安全威胁
  - 权限滥用、信息泄露、代码注入、钓鱼
  - 动态识别VA：DIPrint
- 恶意VA给恶意软件带来了一种新颖的传播方式
  - VA占坑和插件加载策略识别恶意VA：VAHunt

谢谢聆听!

傅建明

jmfu@whu.edu.cn