第 61 卷 第 1 期
2015 年 2 月

武汉大学学报(理学版)
J. Wuhan Univ. (Nat. Sci. Ed.)

Vol. 61 No. 1
Feb. 2015,021～033

综 述

DOI:10.14188/j.1671-8836.2015.01.003

Android 恶意软件检测研究与进展

彭国军[1,2,3],李晶雯[1],孙润康[1],肖云倡[1]

(1.武汉大学 计算机学院,湖北 武汉 430072;
2.空天信息安全与可信计算教育部重点实验室,湖北 武汉 430072;
3.中国人民大学 法学院,北京 100872)

摘 要:针对持续恶化的 Android 安全形势,从恶意软件检测的角度,首先总结了 Android 恶意软件在安装、触发和恶意负载方面的特征和发展趋势;以此为基础,结合 Android 平台特性和移动智能终端环境限制,系统化论述了现有 Android 恶意软件分析与判定技术,指出了权限分析、动态分析和静态分析的实现方法及其优缺点;介绍了基于特征值和基于启发式的恶意软件判定方法.最后,根据已有 Android 恶意软件检测研究的不足,提出了未来的研究方向和发展趋势.

关 键 词:Android;恶意软件特征;恶意软件分析;恶意软件检测

中图分类号:TP 393　　**文献标识码**:A　　**文章编号**:1671-8836(2015)01-0021-13

Android Malware Detection Research and Development

PENG Guojun[1,2,3], LI Jingwen[1], SUN Runkang[1], XIAO Yunchang[1]

(1. School of Computer, Wuhan University, Wuhan 430072, Hubei, China;
2. Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Eduction, Wuhan University, Wuhan 430072, Hubei, China;
3. Law School, Remin University of China, Beijing 100872, China)

Abstract: In light of deteriorating security situation on the popular Android platform, there is a pressing need to develop effective solutions of Android malware detection. To address this issue, firstly Android malwares were systematically characterized from various aspects, including their installation methods, activation mechanisms as well as the nature of carried malicious payloads. Characterization and evolution of Android malware were the foundation of malware detection. Given the resource-constrained mobile phone environments and Android's special programming paradigm, several analysis techniques for detecting malware have been proposed: permission analysis, static analysis and dynamic analysis. The following malware detection solutions are primarily implemented using two methods: signature-based and heuristic-based. And then, a wide range of Android-based malware detection works were compared to evaluate the effectiveness of techniques for analyzing and identifying mobile malware. Finally, the direction of future studies in this field was presented on the basis of the assessment of the previous researches.

Key words: Android; malware characterization; malware analysis; malware detection

0 引 言

2007 年至今的短短七年时间里,Android 一跃成为市场占有率最高的移动智能终端操作系统.生态封闭的 iOS 和 Windows Phone 保证了开发平台和产品一致性,应用程序经过严格审查,价值链控制严密.而 Android 系统秉持开放性原则,吸引了大批用户、手机厂商、应用开发商和销售商[1],但是开放

收稿日期:2014-07-09
基金项目:国家自然科学基金(61202387,61202385,61373168, 61103220);中国博士后科学基金(2012M510641);高等学校博士点专项科研基金(20120141110002),湖北省自然科学基金(2011CDB456)资助项目;武汉市晨光计划项目(2012710367)
作者简介:彭国军,男,副教授,博士,现从事恶意软件检测、移动智能终端安全、电子证据方向的研究. E-mail: guojpeng@whu.edu.cn

的应用市场和相对宽松的应用审查机制[1]导致Android恶意软件泛滥、安全威胁与日俱增.

现有Android恶意软件反逆向、抗检测技术不断发展成熟,恶意软件变种迅速,家族种类繁多,已经形成了以吸费扣费、恶意推广、隐私贩卖为目的的灰色利益产业链.安全公司每天面临大量待测可疑样本,分析检测工作量巨大,特征提取效率低下,导致了基于静态扫描引擎的安全软件检出率不高.应用市场间竞争激烈,同质化严重,基本处于无序发展状态.这样的"内忧外患"恶化了Android安全形势.

受电池、带宽、CPU、内存资源限制,传统适用于PC的恶意软件分析检测手段不能直接运用在移动智能终端上.如何针对Android移动平台恶意软件进行快速有效的分析检测成为当务之急.

现有Android恶意软件检测方面研究普遍侧重恶意软件分析和判定技术的实现原理描述和效果评估,缺乏系统化的总结和对比.

本文由5个部分组成,第1节剖析现有Android恶意软件在安装、功能触发和恶意负载方面的特征和恶意软件发展情况;第2节结合Android平台特性从权限分析、动态分析和静态分析的角度,系统论述现有Android恶意软件分析技术及其优缺点;第3节讨论恶意软件判定方法,即基于特征值和启发式的方法;第4节针对现有研究中的不足之处,提出了几点未来的发展方向.

1 恶意软件特征

本节将从安装、功能触发和恶意负载三个方面描述现有恶意软件特征,为后文恶意软件分析和判定奠定基础.

1.1 安装方式

Android恶意软件往往伪装成热门应用,吸引用户下载,达到快速传播的目的.恶意软件进入用户手机的安装方式,可归纳为重打包、更新包方式和偷渡式下载.这些安装方式可被叠加使用.

1)重打包:选择流行应用程序进行反编译,植入恶意负载,然后重新编译并提交新应用给官方或第三方市场.该方式还能隐藏恶意负载,如以合法安全的方式命名(例如AnserveBot[2]负载的包名为com.sec.android.provider.drm)和代码混淆技术、运行时解密等反静态分析技术.

2)更新包:重新打包流行应用程序,不同于重打包方式在apk文件中存储恶意负载,更新包方式在运行时动态获取或下载恶意负载.静态扫描此类重打包程序无法捕获恶意负载,如BaseBridge[3]、DroidKungFuUpdate[4]、AnserverBot和Plankton[5]等恶意软件使用此方法.

3)偷渡式下载:引诱用户访问恶意网站在未经用户允许的情况下下载安装伪装的恶意软件.

4)其他方式:间谍软件(社会工程学方法安装)、伪装应用(没有实现伪装应用的功能)、山寨应用(实现了山寨对象的功能,同时包含一些隐蔽的恶意功能).

1.2 触发方式

恶意软件被安装后通过下列两种方式被触发启动执行:

1)诱导用户点击运行

重打包应用或伪装应用可被用户直接点击,触发运行.

2)监听系统事件

通过监听相关系统事件,广播机制会自动进行事件通知和函数回调.因此恶意软件可以不依靠用户点击启动应用而灵活触发和运行恶意负载.

例如BOOT_COMPLETED常被恶意软件用于开机自启动或开启后台服务,SMS_RECEIVED被吸费扣费类恶意软件用于监听拦截来自网络运营商的服务定制确认短信和话费使用短信.也有恶意软件(如AnserverBot)通过注册多种监听事件达到可靠、快速启动携带负载的目的.

1.3 恶意负载

Android恶意软件特征更多地体现在其恶意负载上,主要有特权提升、远程控制、话费吸取、隐私窃取和自我保护这5类.

1.3.1 特权提升

特权提升模块能够突破Android权限机制和沙箱机制,允许恶意软件执行特权操作.已知Android平台漏洞及相应root攻击程序如表1所示.

针对Android版本碎片化的问题,恶意软件使用多种root攻击程序,针对不同Android版本,以提高获取root权限的成功率.恶意软件一开始直接使用root攻击程序,例如DroidDream保留了调试输出字符串和root攻击程序文件名;随后引入了加密技术和代码混淆技术,大幅度增加了静态代码分析难度,DroidKongFu加密root攻击程序并存储在resource或asset中,DroidCoupon和GingerMaster混淆root攻击程序相关文件名,伪装成png后缀的图片文件.

表 1　可利用 Android 平台的漏洞及 root 攻击程序列表

漏洞利用对象	root 攻击程序	发布时间	使用该 Exploit 的恶意软件
Linux kernel	Asroot[6]	2009.08.16	Asroot
init(<=2.2)	Exploid[7]	2010.07.15	DroidDream，zHash，DroidKungFu
adbd(<=2.2.1)	RATC[8]	2010.08.21	DroidDream，BaseBridge，DroidKungFu，
zygote(<=2.2.1)	Zimperlich[9]	2011.02.24	DroidDelux，DroidCoupon
ashmen(<=2.3.1)	KillingInTheNameOf[10]	2011.04.21	
vold(<=2.3.3)	GingerBreak[11]	2011.04.21	GingerMaster
libsysutils	zergRush[12]	2011.10.10	
PowerVR SGX (2.3.5)	Levitator[13]	2011.11.03	
memwrite (<=4.0.4)	Mempodroid[14]	2012.01.23	
adb restore (4.0，4.1)	Bin4ry[15]	2012.09.14	

特权提升除了 root 设备使恶意软件以 root 权限运行外，还可以利用存在缺陷的应用程序进行混淆代理人攻击(confused deputy attack)或合谋攻击(collusion attack)造成权限泄露或传递，达到特权提升的目的.

1.3.2　远程控制

远程控制模块被用于远程灵活操控，负责信息回传、更新本地恶意功能.

多数恶意软件远程控制模块使用 http 协议接收 C&C(command-and-control)服务器的控制指令，体现了恶意软件的网络特性. 通过加密本地远程 C&C 服务器 URL 和加密通信过程，增强隐蔽性. 例如 Pjapps 使用自定义的编码方式加密 C&C 服务器地址，Geinimi 使用 DES 加密与 C&C 服务器的通信过程. C&C 服务器种类也从攻击者控制的域服务器，发展到云服务器或者公共 blog 服务器.

1.3.3　话费吸取

话费吸取模块瞄准手机用户的资费，强行定制 SP(service provider)服务并从中牟利. 恶意软件还会过滤通信服务商的短信，达到秘密扣费的目的. 增值服务商号码可以以硬编码方式存储在本地，或者更隐蔽地从远程控制服务器动态获取.

1.3.4　隐私窃取

隐私窃取模块可以实现短信、通讯录和通话记录的获取，通话录音和背景录音，定位，拍照等功能，收集用户隐私数据、社交数据和设备数据. 移动设备隐私信息具有多方面的价值，一旦被获取将用于进一步的攻击. 例如，通过地理位置、社交信息和连接信息精准定位确定目标；将搜集信息用于更“可信”的社会工程学攻击和信任链欺诈；对目标网络和 PC 设备便捷渗透，移动设备可以作为新的“摆渡”渠道，更稳定得传输指令和数据；通过收集移动设备平台类型和软件环境，结合漏洞利用绕过各类安全检测，或利用地理位置和远程控制的条件性发作，达到保持隐蔽的目的. 尤其在 APT(Advanced Persistent Threat，高级可持续威胁)攻击趋势下，信息泄露不容小觑.

1.3.5　自我保护

Android 恶意软件开始采用商业级代码保护技术，造成自动化静态分析失败，人工反汇编反编译困难，启发式检测、已知特征检测等难度加大. 免费工具 ProGuard[16] 可用于压缩、优化和混淆 Java 字节码文件，支持符号信息混淆(Identifier Mangling)，能够轻度对抗检测，被恶意软件误用将增加分析难度. 木马 Obad[17] 利用工具 DexGuard(ProGuard 商业版)实现了 Manifest 字段混淆和基于 clinit 的动态代码解密，被称为“史上最强 Android 木马”. Tim 在 2012 黑帽大会上提出了 Dex 头隐藏代码的技巧[18]，随后被在线加固服务 HoseDex2Jar 采用，而在 2013 年 6 月 Syrup 家族就出现了这一特征. ADAM[19] 是一个 Android app 自动重打包混淆框架，用于测试反病毒软件对抗混淆的检测能力，随后安全公司样本库中就出现了大量被“ADAM 化”的恶意软件.

恶意软件也会挖掘利用 Android 系统、在线服务、预装应用和第三方应用中的漏洞，达到自我保护的目的. 如 Obad 利用系统管理器漏洞注册为系统管理器防止卸载，Skullkey 使用 Android Master Key 的 APK 签名漏洞对正常应用重打包，不破坏正常软件签名，极具隐蔽性. Android 庞大的系统、框架和组件中持续出现漏洞利用的安全问题，它们会影响几乎所有上层应用软件的数据、业务和代码安全. 在版本碎片化的现状下，漏洞利用的问题在部分版本中可能长期得不到修复.

恶意软件特征及其发展趋势表明，恶意软件技术发展迅速，具有快速的自我进化能力. 攻击者互相学习借鉴混淆、反调试、反分析检测技术，构成多层次多角度的安全威胁.

2 恶意软件分析

恶意软件分析,可用于恶意软件检测和判定,分析恶意软件和攻击者的特征、模式和行为,帮助识别网络攻击的真正动机和意图,预测新的安全威胁及其发展趋势,从而生成对进一步保护终端和用户有价值的信息.

下面将从权限分析、动态分析和静态分析的三个方面论述恶意软件分析技术的实现及其优缺点.

2.1 权限分析

Android系统利用权限机制保护敏感API,应用程序根据自身提供的功能,要求合理的权限.分析应用所需权限,能够简单判定这款应用是否安全.

Enck[20]利用Android权限分析识别应用的危险功能.Kirin将危险功能分解为运行所需权限组合,对程序安装时的权限声明进行认证.通过定义不能同时申请的权限组合,与应用程序声明的功能比对,检测具有潜在威胁的应用程序,拒绝恶意软件安装.例如PHONE_STATE、RECORD_AUDIO和INTERNET的权限组合不能被同时申请,因为缺少任意权限都无法进行通话录音这类隐私窃取行为,若图片浏览类应用具有这一行为就会被认为具有潜在威胁.

文献[21]评价Android权限机制的有效性,发现应用程序存在过度申请权限的情况.文献[22]构建了Android API和权限之间的对应关系,提出了Stowaway工具用于静态分析检测应用程序中权限过度申请的情况.

通过分析敏感行为对应权限,类比正常和恶意软件使用权限特征,利用异常检测,可过滤筛选出具有恶意行为的潜在危险应用程序或伪装应用,供下一步人工分析,从而提高安全分析性能和效率.但是恶意软件可以通过特权提升不申请敏感权限而绕过权限机制,以及过度申请权限的问题,这些都将导致直接使用权限信息的检测方法误报率高、可用性差.

Android权限设置粒度过粗,也为权限分析带来了不确定性.权限分析本身具有片面性和局限性,需要结合静态分析和动态分析技术才能做出进一步的判定.由于Android并非利用系统调用实现应用隔离,系统调用完整呈现应用行为存在困难.Zhang[23]利用Android基于权限的隔离机制,从权限使用角度自动化高效构建敏感行为,包括应用程序如何使用受权限控制的Android API访问系统敏感资源,以及权限保护的资源如何被利用,用于检测信息泄露.论文实验证明VetDroid清晰重建了细粒度的恶意行为,相比TaintDroid能够发掘更多信息泄露.

2.2 动态分析

动态分析关注程序的实际行为,因此动态分析不受代码混淆技术影响.表2列出了部分Android恶意软件动态分析系统及其特征提取方式、行为触发方式和检测方法.

Android动态分析环境分为物理设备和Android模拟器两种.

2.2.1 动态分析对象

动态分析方法一般监测系统调用、网络访问、文件和内存修改,信息的访问模式和处理行为等.

早期多数异常检测技术缺乏Android恶意软件行为模式,将电池消耗作为判定的主要依据.通过监控手机能量消耗,类比正常能量消耗模型,检测异常.例如,2004年的Battery-Based Intrusion Detection System(B-BID)[36]和2008年的Battery-Sens-

表2 Android恶意软件检测系统的动态分析技术及其检测方法

系统	特征提取方式			行为触发方式				检测方法	
	API	syscall	VMI	人工	Fuzz	event	UI	特征	启发式
pBMDS[24] 2010.3	-	√	-	√	-	-	-	-	Hidden Markov Model
Shabtai等人[25] 2010.8	√	√	-	-	-	-	-	-	KBTA
AASandbox[26] 2010.10	-	√	-	-	-	-	-	ClamAV	-
PA[27] 2010.11	-	√	-	-	√	-	-	-	-
Crowdroid[28] 2011.10	-	√	-	√	-	-	-	-	k-means聚类
Andromaly[29] 2012.2	√	√	-	-	-	-	-	-	多种分类器
RobotDroid[30] 2012.4	√	-	√	√	-	-	-	-	分类器、主动学习
DroidScope[31] 2012.8	-	-	-	-	-	-	-	-	-
SmartDroid[32] 2012.10	√	-	-	-	-	-	√	-	-
Playground[33] 2013.2	√	√	-	-	√	√	√	-	-
CopperDroid[34] 2013.6	-	-	√	-	-	√	-	-	-
Shabtai等人[35] 2014.2	√	-	-	√	-	-	-	-	概率

ing Intrusion Protection System (B-SIPS)[37]系统. 这类检测仅对消耗手机资源的恶意软件有效，将恶意功能推迟到充电时执行将绕过这类检测方法.

以 Linux 相关安全研究为基础，动态分析开始关注内核层的系统调用(system call，简写 syscall). 监控系统调用被认为是最准确的判断程序行为的方式之一，但是系统调用属于底层信息. 随后又发展出了上层的 API 调用分析、信息流跟踪和网络流量监控，以全面掌握程序运行状态，恶意软件判定结果也更准确. 图 1 结合 Android 系统层次给出了 API 调用与系统调用的关系.

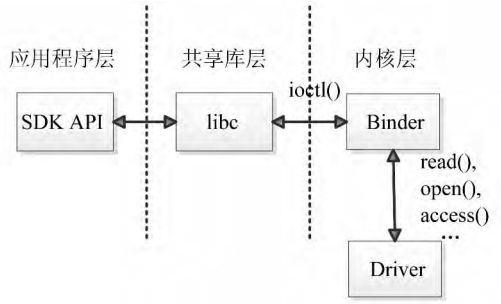

图 1　API 调用与系统调用关系

1) 系统调用(内核层)

系统调用能够体现应用程序和底层系统之间的交互特征，不同研究的区别仅在于如何构建系统调用集合[38](系统调用向量[29]，系统调用序列和调用参数[39]，系统调用时序模式[40]，资源访问模型[41]，系统调用依赖图[41~43]).

系统调用收集到的程序执行信息来自底层，结果准确且不易被干扰和绕过，但是缺少上层调用的语义信息，造成这种局限性的原因如下：

① Android 应用程序框架层管理资源：Android 系统资源大多由应用程序框架层管理和保护，应用与系统的交互(比如访问联系人，通话记录)发生在比系统 Linux 内核级调用更高的语义级别. 使用系统调用了解应用程序与 Android 的交互将失去资源访问的语义，降低了重建行为的质量和精度.

② Binder 机制进行进程间通信：利用进程间通信，单个程序执行流程可被分割成多个进程联合执行，从而绕过系统调用行为监控[44]，因此仅拦截与 Binder 驱动交互的系统调用，无法获得应用执行行为的真实完整信息.

③ 事件触发：Android 广播机制是异步资源访问机制，注册的回调函数在应用程序层，因而基于广播机制的资源访问行为能够绕过系统函数监控.

文献[45]提出了监控 Linux 内核事件的方法，在分析 Linux 安全性增强类工具的基础上，从内核提取系统调用、文件修改等操作，生成正常行为模型. 但是当时还没有真正的 Android 设备. Crowdroid[28]、AASandbox[26]等系统在设备上收集系统调用的行为特征. Playground[33]利用系统调用和内核数据结构检测 root 利用攻击程序.

2) API 调用(应用程序框架层)

Android SDK 的 API 实现在应用程序框架层. 动态分析关注敏感行为类 API(例如发送短信 API 及其发送号码和内容)、Java 反射类 API 及其方法调用和域访问、动态字节码加载类 API 等. 通过解析 API 调用语义，同样能够还原出程序运行轨迹. 缺点是容易被绕过，不够全面和准确.

AppProfiler[46]提出两步式转义技术，将 API 调用映射为高层次行为描述. Playground[33]、Andromaly[29]、SmartDroid[32]都是通过监控敏感 API 调用检测恶意功能. 特别地，Shabtai 等人[35]利用网络通信相关 API 提取软件网络特征，识别更新包方式安装的恶意软件.

3) 资源访问(应用程序框架层)

资源访问和资源使用模式可用于重建恶意软件行为. 结合污点跟踪技术，获得跟踪资源使用信息能够细粒度地检测信息泄露甚至漏洞利用. 例如 Android 恶意软件常用的串谋攻击和混淆代理人攻击. VetDroid[23]动态跟踪资源使用权限请求和对授权资源的敏感操作，分析应用程序利用权限访问系统敏感资源的过程.

4) 网络特征(应用程序框架层或内核层)

更新或下载恶意负载、与远程控制服务器通信、回传隐私数据等网络行为是恶意软件行为的重要组成部分，通过识别网络流量模式改变，分析流量特征变化原因，辨别是由用户操作或应用更新等正常行为还是恶意攻击造成的. 现有 Android 平台网络特征分析[35]较少，主要还是受系统性能和开销限制，未来可能会从内核层捕获网络数据包，通过逆向网络协议分析网络负载内容，用于网络协议安全性检测和入侵检测/异常检测.

2.2.2　行为触发

行为触发主要有人工触发和模拟触发，其中模拟触发又包括随机触发/模糊测试、事件触发和 UI 触发.

1) 人工触发：测试人员根据策略运行恶意软件或远程收集普通用户使用情况；

2) 模拟触发：使用工具向系统发送伪随机的用户事件流(如按键输入、触摸屏输入、手势输入等)，这类测试工具有 Monkey 和 Money runner、Junit 和

IRobotium.

① 随机触发/模糊测试(fuzzing test):随机生成用户模拟点击、滑动、输入等操作,由于模拟操作不含语义或上下文信息,如果设计得当理论上能够完全遍历待测软件的有限状态空间,但是模糊测试无法满足某些软件设置的检查条件;

② 事件触发:根据 Android 广播机制和软件 AndroidManifest.xml 中声明的监听事件类型,构造该类型事件并触发广播;

③ UI 触发:根据预先定义的策略,解析上下文信息和语义信息,如用户登录界面中的用户名和密码输入框,然后模拟用户点击、滑动、输入等操作.

Google Bouncer 实现了自动遍历触发和检测,但是不能应对伪装技术和反模拟器技术[47,48]. Smartdroid[32]提出自动化 UI 遍历的检测框架,利用静态分析技术生成敏感行为相关的活动调用图(Activity Call Graph, ACG)和函数调用图(Funtion Call Graph, FCG)从程序入口 Activity 开始自动化 GUI 测试. Playground[33]使用多种触发方式,使用模糊测试简单高效地触发,同时配合自动遍历技术感知界面元素和上下文信息进行事件触发和 UI 触发. Gilbert 等人[49]讨论当模糊测试方法生成的随机输入遇到一些 UI 限制而陷入僵局时,提出了符号执行(symbolic execution)和实值执行(concrete execution)相结合的方法,利用符号执行得到相应的路径约束,再解析出触发目标代码的具体值进行实值执行,进行 UI 触发.

因此行为触发是动态分析的重点和难点,旨在全面暴露出恶意软件功能和行为轨迹. 行为触发存在下列固有不足:

1) 很难模拟出合适的环境触发恶意软件全部功能;

2) 无法确定观测恶意软件行为触发所需的时间长度.

2.2.3 动态分析方法

1) 特征提取和行为分析

现有恶意软件行为提取采用 smali hook、Binder IPC 注入、ROM 定制、Zygote 注入和虚拟机自省(VM Introspection, VMI)方式.

① Smali hook:修改 APK 反编译后的 smali 代码,插入监控模块,但是反编译对抗技术可使 APK 文件无法成功反编译,导致这类方法失效.

② ROM 定制:修改 Android 源代码插入监控模块重新编译 ROM 或使用内核可加载模块实现监控,可监控系统调用、API 调用、资源访问和网络行为,实现方法简单使用广泛.

③ Binder IPC 注入:利用进程间通信机制 Binder,注入关键函数 ioctl()获取进程敏感行为调用的信息,Peng 等人[50]使用这种注入方式进行敏感行为监控和恶意软件检测.

④ Zygote 注入:利用 Android Zygote 进程创建新进程的原理,注入并监控应用程序框架层 API 调用, xposed 框架[51]开源项目采用此类技术.

⑤ 虚拟机自省:基于 hypervisor 的虚拟化安全解决方案,从 Android 模拟器外部监控系统和应用程序状态,优点是在虚拟机外部进行分析,不受监控层次约束,能够捕捉到内核最高特权攻击,保证外部分析策略不受内部攻击干扰;但是,缺乏上下文语义信息,为了重建语义信息,VMI 需要监控特定的内核事件并扫描内核数据结构;DroidScope[31]和 CopperDroid [34]均采用了 VMI 方式.

2) 语义分析

行为分析从识别单一行为,发展为解析行为语义,提高了恶意软件分析和检测的准确性. Android 拥有 OS 层次和 Java 层次的语义信息. OS 层次可了解恶意软件进程及其 native 组件的活动;Java 层次语义可掌握 Java 组件行为.

为了捕获 Java 和 native 组件交互,Shabtai 等人[25,29]使用基于知识的时序抽象(knowledge-based, temporal abstraction, KBTA)方法,检测恶意行为的可疑时序模式,对 DoS 或蠕虫等连续攻击行为检测效果较好. Droidscope[31]从系统调用和 Java调用中同步构建 OS 层和 Java 层的行为语义.

用户意图是对行为语义的进一步抽象和提取. 智能手机应用以用户为中心,采用交互密集型设计模式,由用户的特定行为触发对应操作. 正常软件中敏感系统操作,例如网络和文件访问,都是直接或间接由用户请求触发的,而恶意软件往往不经过用户明确的认可而滥用数据和系统资源. 将用户意图与自动化可度量的程序分析结合,可用于行为恶意性识别和可信评估.

3) 污点跟踪

污点跟踪是一种可用于动态监控程序运行并记录程序中数据传播的技术. 污点跟踪包括确定污点源、跟踪污点传播路径和污点出口捕获.

TaintDroid[52]较早使用动态污点分析技术,首先标记由源头 API(涉及位置、摄像头、麦克风和手机标识符操作的 API)产生的数据;然后实现细粒度的 Dalvik VM 指令级别和粗粒度的文件和 IPC 消息级别的污点跟踪;最后在污点出口(网络通信 API

中使用的 data buffer)侦查污点标记.

受污点出口定义的限制，TaintDroid 只能检测到传出手机的隐私行为，而不能检测全部的隐私窃取行为. 从污点跟踪实现角度，TaintDroid 只跟踪显式数据流，没有使用控制流分析，从而无法检测如 Soundcomber[53] 这类利用隐蔽信道(手机震动、选项设置等)的攻击，但这类行为可利用静态分析识别. 污点跟踪技术的提出促使黑客隐藏其隐私窃取回传行为. 从实现层次来看，TaintDroid 只针对 Dalvik bytecode，不包含 native code 的污点跟踪，后者需要动态二进制替换和测试覆盖插入(Binary instrumentation)或虚拟机自省，可能会大量消耗系统资源.

MockDroid[54] 和 TISSA[55] 基于 TaintDroid 的污点跟踪框架，用于验证系统有效性. AppFence[56] 在 TaintDroid 的基础上提供更细粒度的系统资源访问控制. Gilbert 等人[49] 通过扩展 TaintDroid 实现隐藏数据流的跟踪. Playground[33] 集成 TaintDroid 使用污点跟踪发现隐私泄露. Appintent[57] 采用污点分析的思路(static taint，dynamic taint)分析敏感数据的传播，将符号执行与 Android 活动生命周期、GUI 事件和系统事件结合生成自动化测试方法，评估用户是否知晓隐私数据传输，以精确判断应用软件是否存在隐私泄露.

污点跟踪的缺点是进行数据流和控制流跟踪时可能面临大量的性能消耗和污点爆炸问题.

动态检测的准确性在于检测点获得信息的准确性，内核层更准确，软件层语义更丰富.

2.3　静态分析

静态分析是指在不运行代码的情况下，采用控制流分析、数据流分析和语义分析等各类技术手段对 APK 文件及其解析生成的文件进行分析的技术.

静态分析的优点是快速、高效，但是难以应对代码混淆和多态变形技术. 同时，静态分析难以识别漏洞利用攻击，因为漏洞利用攻击的分析和定位需要具体的漏洞运行环境.

部分静态分析系统特征及其使用静态分析方法如表 3 所示.

表 3　静态分析系统特征对比

系统	目的	静态分析方法	判定依据	论文实验样本规模
ScanDroid[58]	实施秘密性和完整性	数据流分析和字符串	权限逻辑的约束	—
CHEX[59]	发掘暴露的组件 API	数据流分析	公开且未作约束的组件	5 486 个应用
SEFA[60]	固件安全	数据流分析、控制流分析	—	10 个手机镜像
Droidchameleon[61]	评估恶意软件对抗重打包能力	字符串和数据流分析、控制流分析	恶意软件行为特征	10 款商用反病毒软件，6 款恶意软件
UID[62]	识别未授权调用	数据流分析、事件相关控制流分析	对特权函数调用的触发与操作之间的依赖关系	708 个应用和 482 个恶意软件
RiskRanker[63]	检测异常代码或行为模式	数据流分析、控制流分析	恶意软件行为特征	118 318 个应用
Woodpecker[64]	固件权限	数据流分析、控制流分析	—	8 个手机镜像，13 个权限
AndroidLeaks[65]	秘密性	数据流分析	危险 API 使用的敏感数据	24 350 个应用
SCANDAL[66]	秘密性	数据流分析	危险 API 使用的敏感数据	90 个应用和 8 个恶意软件
Stowaway[22]	检测过度申请权限应用	字符串和 Intent 控制流分析	比较需要和申请权限	940 个应用
ComDroid[67]	检测应用程序间通信漏洞	Intent 控制流分析	弱权限或没有权限限制的隐式 Intent	100 应用

2.3.1　静态分析对象

APK 文件结构包括 classes.dex、AndroidManifest.xml 和资源文件夹等，其中 classes.dex 是 Java 文件编译生成的 Dalvik 字节码，资源文件夹存放可执行的共享库文件和 dex 文件等. 这里从 APK 文件编译的角度划分静态分析对象，图 2 给出了现有检测系统关注的静态分析对象类型.

1) Java 源代码：Java 源代码可由 Dalvik 字节

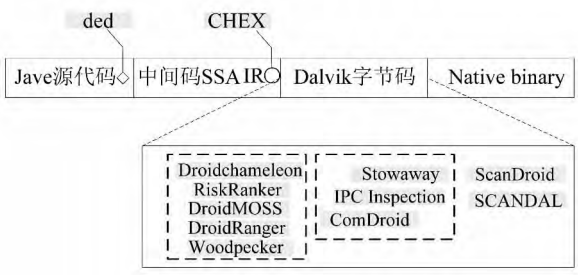

图2 静态分析对象与代表性系统

码反编译而成,在反编译过程中可能会引入歧义或错误;

2) Dalvik字节码:Dalvik字节码包含高层指令的特点,可提取指令语义信息,便于生成特征模板,因而被广泛采用,但是研究表明字节码随机化技术能够完全隐藏Dalvik字节码内部逻辑[68];

3) Native Binary:不适用于语义检测,需要动态二进制替换和测试覆盖插入(Binary instrumentation)或VMI,会遇到函数边界确定困难、指针混淆等问题;

4) 其他对象:CHEX[59]将Dalvik字节代码转换为SSA IR (static single assignment,SSA,基于语义的转义)使用DexLib的Dalvik字节码解析器,它提供了读取字节码中嵌入数据、类型信息和Dalvik指令的编程接口.

2.3.2 静态分析入口点

为了快速定位Android程序中的关键代码,需要确定程序入口.

虽然Android应用程序在AndroidManifest.xml文件中定义了动作android.intent.action.MAIN用于指示了应用程序的入口点.但是Android的广播机制是基于事件驱动或基于回调的,当注册监听的事件发生时,Android广播机制会自动启动对应的监听响应组件.因此,Android中存在多种入口点类型.

CHEX[59]采用基于启发式的方法发现应用程序所有可能的入口点,误报率也相对较低.CHEX引入SDS(split data-flow summary)模型,排列整合数据流各分支,对异步调用进行建模,提取并导出数据流间的行为.为了给多入口点的执行过程建模,文献[69]通过hook main函数模拟GUI系统中的事件循环调度程序(event loop dispatcher),依赖特定的域知识来检测常见组件的入口点,但是无法保证检测的完整性.与文献[69]相比,CHEX将整个数据流分析问题分为很多规模较小且自包含的子问题,这种松耦合结构提高了性能和可扩展性.

2.3.3 静态分析方法

静态分析的主流分析手段如下,对应的系统使用情况已在表4中列出.

1) 控制流分析:生成有向控制流图,用节点表示基本代码块,节点间的有向边代表控制流路径,反向边表示可能存在的循环;或生成函数调用关系图,表示函数间的嵌套关系.通过预先定义异常控制流路径发现程序异常行为,例如包含数据的Intent在没有保护的情况下发送给匿名目标,这一执行路径将带来潜在的信息泄露.

2) 数据流分析:对控制流图进行遍历,记录变量的初始化点和引用点,保存相关数据信息.

3) 语义分析:语义分析可以审查程序执行的上下文有关性质.例如语义分析发现Android短信发送方法sendTextMessage()的第一个参数为常量时,认为短信发送给了一个硬编码的电话号码.

如果在静态分析中仅利用系统调用代表程序行为,而忽略了调用序列的语义,容易被混淆或伪装而绕过.Christodorescu等人[70]提出了基于静态语义的恶意软件检测方法,试图检测代码混淆,以识别恶意软件变种中语义相近的序列.在被反汇编的二进制代码上进行匹配算法,以发现符合恶意行为模板中定义的指令序列,例如解密操作的循环等.抽象化寄存器和符号常量名称能够适应一些代码混淆技术.Droidchameleon[61]则利用了现有Android恶意软件代码混淆技术的代码特征,用于评估现有安全软件对抗代码混淆的能力.但是,这类方法需要应用程序指令和模板完全匹配,当攻击使用指令替换或者重新排序仍然有可能绕过.尽管基于语义的特征检查方法提高了对恶意软件变种的检出率[70],但是基于特征的方法无法查杀未知恶意软件.

文献[71]系统介绍了Android安全机制、进程间通信和组件间通信,其中基于Intent的组件间通信由于未对信息流进行监控,Android系统存在潜在的安全威胁:一方面,Intent传递的消息可以被嗅探、窃取和篡改,造成隐私泄露;另一方面,恶意软件能够伪造或注入恶意Intent消息污染用户数据,绕过应用的安全策略.ComDroid和Stowaway都采用了基于Intent的控制流分析,结合函数调用和组件调用,挖掘由隐式Intent导致的组件劫持、信息泄漏和权限泄漏,但是这种静态分析方法仅能证明存在漏洞,很难验证是否存在攻击,且受到Java反射机制和动态加载的干扰.

Zhou等人对Android恶意软件特征及其发展进行了详细分析[72],收集并公开了2011到2012年

间 Android 市场上 1260 个恶意软件样本，覆盖了当时 49 个不同的恶意软件家族，恶意软件的特征提取和发展分析启发了后续恶意软件分析和检测工作的开展. 尤其是在应用市场审查方面，静态分析检测主要基于现有恶意软件特征，1）DroidMOSS[73] 针对重打包应用使用模糊哈希（fuzzing hash）的方法检测；2）RiskRanker[63] 检测 root 利用、敏感行为和动态加载加解密行为；3）UID[62] 工具遍历 Java 字节码，生成上下文敏感的过程间和过程内数据流依赖图，以及基于事件的信息流，处理 Android Intents 和 GUI 相关隐式方法调用之间的数据流，计算应用程序中用户输入触发敏感函数调用的比例生成其可信度值；4）Woodpecker[64] 和 SEFA[60] 以手机固件作为安全研究对象，是对现有 Android 软件生态系统安全性评估和构架的重要组成部分.

除此以外，还有一些 Android 平台分析工具值得关注，可用于提高恶意软件分析效率. Androguard[74] 提供了一组工具包辅助分析人员快速鉴别与分析 APK 文件，每个工具都是独立的 py 文件，从查看 apk 文件到生成 Java 方法级与 Dalvik 指令级的图形文件等，方便分析人员快速了解程序的执行流程. Santoku[75] 是一款集成分析环境，继承了大量主流 Android 分析工具、移动设备取证工具、渗透测试工具盒网络数据分析工具.

3　恶意软件判定

在静态和动态分析的基础上，一般采用基于特征和基于启发式这两种方式实现 Android 恶意软件检测.

3.1　基于特征值的判定方法

现有反病毒产品普遍采用基于特征值的精确检测方法. 这种方法要求安全厂商不断收集和分析新的恶意软件样本，提取唯一标识特征加入特征数据库.

基于特征值的精确检测方法能够快速准确地识别并清除已知恶意软件. 缺点是滞后性，即无法检测未知恶意软件.

3.2　基于启发式的判定方法

启发式检测是通过分析指令出现的顺序或特定组合情况等常见恶意软件标准特征来判断是否为恶意软件. 启发式检测有两种方式，动态启发和静态启发.

1）动态启发：在系统中设置若干行为特征监测点实时监控程序行为，通过识别恶意行为以侦测恶意软件. 监控行为对象的确认和行为特征的提取需要以大量恶意软件分析为基础. 监控的对象可以是某一种行为或多种行为的组合. 但是动态启发式可能面临程序行为跟踪深度、行为跟踪粒度等问题，将会导致检测速度缓慢、效率低下.

2）静态启发：对 APK/JAR 文件、APK 反编译后生成的 Smali 代码、JAVA/C/C＋＋源代码的分析来检测恶意软件，通过对文件外部静态信息分类，模拟跟踪代码执行流程来判断是否为恶意软件. 静态启发式较动态启发式效率更高，但是检出率相对较低.

启发式检测的缺点主要集中在误报和效率低这两方面. 为了达到更高的检出率，规则需要设置的更为灵活，而高检出率必然会导致一定程度上的误报. 为避免高的误报，通常可以采用白名单或数字签名验证等技术来消除.

其中，基于规则的检测方法[20] 只能检测符合预先定义的规则库的恶意软件，无法检测未被规则定义的已知恶意软件和不符合规则的未知恶意软件. 异常检测通过定义一组系统处于“正常”情况时的数据，如电量消耗、软件资源访问行为、网络行为、用户操作与软件行为等，然后分析确定是否出现异常. 常见的异常检测技术有基于概率、统计学、人工神经网络、模糊识别和人工免疫学的方法. Andromaly[29] 从分类器选择、特征选择方法和监控特征数量三方面，考察适合 Android 的基于分类器的异常检测方法. 实验采用 Chi-Square、Fisher Score 和 Information Gain 这 3 个特征分类方法，量化每一维特征对分类任务的影响，然后评估了包括 k 均值、逻辑回归、直方图、决策树、贝叶斯网络和朴素贝叶斯的分类器性能和特征选择算法得到特征个数对分类结果的影响.

样本数量是机器学习准确率提升的保障，但鉴于目前 Android 恶意样本的获取难度和分类准确性的问题，适合少量样本、准确率高、开销小的 SVM 算法被广泛采用[39,76].

4　趋势与展望

Android 平台软件生态系统的安全维护需要 Android 系统研发、应用开发者、应用市场、用户使用等多环节群策群力. 目前，国内外在 Android 恶意软件分析与检测方面的研究已经取得了较多理论和实践成果，但还是存在以下不足之处：

1）缺乏权威的检测样本集：当前各类恶意软件检测研究中，普遍缺乏权威的恶意软件样本集合和

规范性分类,样本来源多样化,造成部分研究工作缺乏权威可对比性,不利于相关研究工作的深入推进.

2)自动化动态分析行为触发的智能化程度不高:

① 代码和UI紧密交互为程序分析和自动化挖掘造成困难,对应用程序UI组件的分析不够深入;

② 与PC不同,Android绝大部分漏洞均与应用的业务相关,缺乏有效的业务数据来驱使自动化动态分析;

③ Android应用程序网络特征提取、网络协议分析和逆向方面的研究不足,缺少网络通信的行为触发和对应用程序网络行为的分析.

3)静态分析方面:代码混淆、加壳和反编译对抗等恶意代码保护技术,可能导致自动化静态分析失败,人工反汇编、反编译困难,缺乏成熟的自动化识别和应对工具,如PC上相对成熟的脱壳技术.

4)基于终端的恶意软件分析检测框架部署困难:通常会修改Android系统框架以进行行为监控,在大范围部署并服务于广大用户方面存在困难,实用性大打折扣,并且这些安全应用大多以root权限运行,root设备会带来潜在安全风险.

5)Android系统碎片化问题:分析检测系统往往只针对一种或特定几种Android系统版本,版本迁移困难,无法应对版本针对性攻击.

综合考虑已有工作的不足和Android平台未来的发展趋势,针对Android恶意软件分析检测的安全研究工作将来可能围绕以下几个方面展开:

1)研究样本集的完善:针对Android恶意样本获取难度和分类准确性的问题,建立并不断完善恶意软件检测的基准样本数据集,为更好地进行恶意软件检测分析研究奠定良好的样本基础与检出效果比对基准.

2)Android系统安全增强:在已有Android安全机制基础上扩展Android系统,克服Android系统开放性所带来的固有安全缺陷,提升其恶意软件抵抗能力.如提供内存扫描机制,扩展动态分析范围;构建安全区域(如增加可信域和不可执行域等),增加攻击难度;完善增强native code和Dalvik code的动态加载机制等.

3)样本分析技术自动化:建立快速稳定的静态与动态自动化分析机制,构建恶意软件快速分析平台与渠道,有效提升恶意软件处理速度,并有利于推动App的安全接入检测与应用市场的安全规范.

4)样本关联分析:从微观角度进行恶意软件家族自动化分类研究,提高样本分析效率,把握恶意软件家族内部衍生规律与演化关系,从宏观角度把握不同家族样本的关联性,剖析样本真实攻击意图与背景.

5)APT中小众样本的定位、获取与深入分析,并针对APT形势下的信息泄露漏洞和隐私搜集行为提出隐私类数据的窃取检测与保护.

6)大数据检测与态势感知:构建基于云结构的快速联动检测机制,实时把握移动平台恶意软件攻击态势,预测近期和未来Android恶意软件发展趋势,从而提前部署防御策略.

Android平台的流行造成了恶意软件的泛滥,也吸引了安全研究人员的广泛关注.本文的综述对Android恶意软件检测研究工作具有一定指导意义.

参考文献:

[1] Steven M P. Contrary to what you've heard, Android is almost impenetrable to malware[EB/OL]. [2014-06-23]. *http://qz.com/131436/contrary-to-what-youve-heard-android-is-almost-impenetrable-to-malware/*.

[2] Zhou Y, Jiang X. An analysis of the AnserverBot trojan[EB/OL]. [2014-06-23]. *http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot_Analysis.pdf*.

[3] F-Secure's Security Labs. Trojan: Android/BaseBridge.A[EB/OL]. [2014-06-23]. *http://www.f-secure.com/v-descs/trojan_android_basebridge.shtml*.

[4] F-Secure's Security Labs. Droid KungFu Utilizes an Update Attack[EB/OL]. (2011-10-05)[2014-06-23]. *http://www.f-secure.com/weblog/archives/00002259.html*.

[5] Jiang X X. Security Alert: New Stealthy Android Spyware-Plankton-Found in Official Android Market [EB/OL]. [2014-06-23]. *http://www.csc.ncsu.edu/faculty/jiang/Plankton/*.

[6] Goegre code. Asroot[EB/OL]. [2012-02-07]. *http://code.google.com/p/flashrec/source/browse/#svn%2Ftrunk%2Fandroid-root*.

[7] SEBASTIAN. Android trickery[EB/OL]. [2014-06-23]. *http://c-skills.blogspot.com/2010/07/android-trickery.html*.

[8] SEBASTIAN. Droid2[EB/OL]. [2014-06-23]. *http://c-skills.blogspot.com/2010/08/droid2.html*.

[9] SEBASTIAN. Zimperlich sources[EB/OL]. [2014-06-23]. *http://c-skills.blogspot.com/2011/02/zimperlich-sources.html*.

[10] SEBASTIAN. adb trickery #2[EB/OL]. [2014-06-23]. *http://c-skills.blogspot.com/2011/01/adb-*

trickery-again.html.

[11] SEBASTIAN. yummy yummy, GingerBreak！[EB/OL]. [2014-06-23]. *http://c-skills.blogspot.com/2011/04/yummy-yummy-gingerbreak.html*.

[12] ieftm. Revolutionary - zergRush local root 2.2/2.3 [EB/OL]. [2014-06-23]. *http://forum.xda-developers.com/showthread.php?t=1296916*.

[13] Jon L, Jon O. Android < 2.3.6 PowerVR SGX Privilege Escalation Exploit[EB/OL]. [2014-06-23]. *https://jon.oberheide.org/files/levitator.c*.

[14] se1000. Mempodroid root[EB/OL]. [2014-06-23]. *http://forum.xda-developers.com/showthread.php?t=1461736*.

[15] bin4rg. Root MANY ANDROID[EB/OL]. [2014-06-23]. *http://forum.xda-developers.com/showthread.php?t=1886460*.

[16] Google Android developer tool. ProGuard[EB/OL]. [2014-06-23]. *http://developer.android.com/tools/help/proguard.html*.

[17] COMOPO. Android OBAD Technical Analysis Paper [EB/OL]. [2014-06-23]. *http://www.comodo.com/resources/Android_OBAD_Tech_Reportv3.pdf*.

[18] Tim S. Dex Education: PracticingSafeDex[EB/OL]. [2014-06-23]. *http://www.strazzere.com/papers/DexEducation-PracticingSafeDex.pdf*.

[19] Zheng M, Lee P P, Lui J C. Adam: An automatic and extensible platform to stress test android anti-virus systems[C]//*Detection of Intrusions and Malware, and Vulnerability Assessment*. Berlin: Springer, 2013: 82-101.

[20] Enck W, Ongtang M, Mcdaniel P. On lightweight mobile phone application certification[C]//*Proceedings of the 16th ACM conference on Computer and communications security*. New York: ACM, 2009: 235-245.

[21] Felt A P, Greenwood K, Wagner D. The effectiveness of application permissions[C]//*Proceedings of the 2nd USENIX conference on Web application development*. California: USENIX Association, 2011: 7.

[22] Felt A P, Chin E, Hanna S, *et al*. Android permissions demystified[C]//*Proceedings of the 18th ACM conference on Computer and communications security*. New York: ACM, 2011: 627-638.

[23] Zhang Y, Yang M, Xu B, *et al*. Vetting undesirable behaviors in android apps with permission use analysis [C]//*Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. New York: ACM, 2013: 611-622.

[24] Xie L, Zhang X, Seifert J P, *et al*. pBMDS: A behavior-based malware detection system for cellphone devices[C]//*Proceedings of the third ACM conference on Wireless network security*. New York: ACM, 2010: 37-48.

[25] Shabtai A, Kanonov U, Elovici Y. Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method[J]. *Journal of Systems and Software*, 2010, **83**(8): 1524-1537.

[26] Blasing T, Batyuk L, Schmidt A D, *et al*. An android application sandbox system for suspicious software detection[C]//*In 5th International Conference on Malicious and Unwanted Software, MALWARE'* 2010. Oakland: IEEE, 2010: 55-62.

[27] Portokalidis G, Homburg P, Anagnostakis K, *et al*. Paranoid Android: Versatile protection for smartphones[C]//*Proceedings of the 26th Annual Computer Security Applications Conference*. New York: ACM, 2010: 347-356.

[28] Burguera I, Zurutuza U, Nadjm-Tehrani S. Crowdroid: Behavior-based malware detection system for android[C]//*Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. New York: ACM, 2011: 15-26.

[29] Shabtai A, Kanonov U, Elovici Y, *et al*. "Andromaly": A behavioral malware detection framework for android devices[J]. *Journal of Intelligent Information Systems*, 2012, **38**(1): 161-190.

[30] Zhao M, Zhang T, Ge F, *et al*. RobotDroid: A lightweight malware detection framework on smartphones [J]. *Journal of Networks*, 2012, **7**(4):715-722.

[31] Yan L K, Yin H. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis[C]//*Proceedings of the 21st USENIX Security Symposium*. California: USENIX Association, 2012:569-584.

[32] Zheng C, Zhu S, Dai S, *et al*. Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications[C]//*Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. New York: ACM, 2012: 93-104.

[33] Rastogi V, Chen Y, Enck W. Appsplayground: Automatic security analysis of smartphone applications [C]//*Proceedings of the third ACM conference on Data and application security and privacy*. New York: ACM, 2013: 209-220.

[34] Reina A, Fattori A, Cavallaro L. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors [DB/OL]. [2013-06-23]. *http://www.cs.swarthmore.edu/~*

bylvsa1/cs97/f13/Papers/copperdroid.pdf.

[35] Shabtai A, Tenenboim-Chekina L, Mimran D, *et al*. Mobile malware detection through analysis of deviations in application network behavior[J]. *Computers & Security*, 2014, **43**: 1-18.

[36] Jacoby G A, Davis Iv N J. Battery-based intrusion detection[C]//*Proceedings of the Global Telecommunications Conference, GLOBECOM'* 04. Oakland: IEEE, 2004: 2250-2255.

[37] Buennemeyer T K, Nelson T M, Clagett L M, *et al*. Mobile device profiling and intrusion detection using smart batteries[C]//*Proceedings of the Proceedings of the* 41*st Annual Hawaii International Conference on System Sciences*. Oakland: IEEE, 2008: 296.

[38] Canali D, Lanzi A, Balzarotti D, *et al*. A quantitative study of accuracy in system call-based malware detection[C]//*Proceedings of the* 2012 *International Symposium on Software Testing and Analysis*. New York:ACM, 2012: 122-132.

[39] Bose A, Hu X, Shin K G, *et al*. Behavioral detection of malware on mobile handsets[C]//*Proceedings of the* 6*th international conference on Mobile systems, applications, and services*. New York: ACM, 2008: 225-238.

[40] Lanzi A, Balzarotti D, Kruegel C, *et al*. AccessMiner: using system-centric models for malware protection [C]//*Proceedings of the* 17*th ACM conference on Computer and communications security*. New York: ACM, 2010: 399-412.

[41] Christodorescu M, Jha S, Kruegel C. Mining specifications of malicious behavior[C]//*Proceedings of the* 1*st India software engineering conference*. New York: ACM, 2008: 5-14.

[42] Comparetti P M, Salvaneschi G, Kirda E, *et al*. Identifying dormant functionality in malware programs [C]//*Proceedings of the* 2010 *IEEE Symposium on Security and Privacy*. Oakland:IEEE, 2010: 61-76.

[43] Fredrikson M, Jha S, Christodorescu M, *et al*. Synthesizing near-optimal malware specifications from suspicious behaviors[C]//*Proceedings of the* 2010 *IEEE Symposium on Security and Privacy*. Oakland:IEEE, 2010: 45-60.

[44] Sparks S, Butler J. Shadow walker: Raising the bar for rootkit detection[J]. *Black Hat Japan*, 2005, **11** (63):504-533.

[45] Schmidt A D, Schmidt H G, Clausen J, *et al*. Enhancing security of linux-based android devices[DB/OL]. [2013-06-23]. *http://www.dai-labor.de/fileadmin/files/publications/lk*2008*-android_security.pdf*.

[46] Rosen S, Qian Z, Mao Z M. Appprofiler: A flexible method of exposing privacy-related behavior in android applications to end users[C]//*Proceedings of the third ACM conference on Data and application security and privacy*. New York: ACM, 2013: 221-232.

[47] Jon O. Dissecting android's bouncer[EB/OL]. [2014-06-23]. *https://blog.duosecurity.com*/2012/06/*dissecting-androids-bouncer/*.

[48] Ryan W. Circumventing google's bouncer, android's anti-malware system[EB/OL]. [2014-06-23]. *http://www.extremetech.com/computing*/130424*-circumventing-googles-bouncer-androids-anti-malware-system*.

[49] Gilbert P, Chun B G, Cox L P, *et al*. Vision: Automated security validation of mobile apps at app markets [C]//*Proceedings of the Second International Workshop on Mobile Cloud Computing and Services*. New York: ACM, 2011: 21-26.

[50] Peng G J, Yuru S, Taige W, *et al*. Research on android malware detection and interception based on behavior monitoring[J]. *Wuhan University Journal of Natural Sciences*, 2012, **17**(5):421-427.

[51] Xposed-ROM modding without modifying APKs[EB/OL]. [2014-06-23]. *http://forum.xda-developers.com/xposed/framework-xposed-rom-modding-mo difying-t*1574401.

[52] Enck W, Gilbert P, Chun B-G, *et al*. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones[J]. *Communications of the ACM*, 2014, **57**(3): 99-106.

[53] Schlegel R, Zhang K, Zhou X-Y, *et al*. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones[C]//*Proceedings of the* 18*th Annual Network and Distributed System Security Symposium, NDSS*'11. Virginia: ISOC, 2011: 17-33.

[54] Beresford A R, Rice A, Skehin N, *et al*. MockDroid: trading privacy for application functionality on smartphones[C]//*Proceedings of the* 12*th workshop on mobile computing systems and applications*. New York: ACM, 2011: 49-54.

[55] Zhou Y, Zhang X, Jiang X, *et al*. Taming information-stealing smartphone applications (on android) [C]//*Proceedings of the* 4*th international conference on Trust and trustworthy computing*. Berlin: Springer, 2011: 93-107.

[56] Hornyack P, Han S, Jung J, *et al*. These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications[C]//*Proceedings of the* 18*th ACM conference on Computer and commu-*

nications security. New York: ACM, 2011: 639-652.

[57] Yang Z, Yang M, Zhang Y, *et al*. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection[C]//*Proceedings of the* 2013 *ACM SIGSAC conference on Computer & Communications Security*. New York: ACM, 2013: 1043-1054.

[58] Fuchs A P, Chaudhuri A, Foster J S. SCanDroid: Automated security certification of Android applications[DB/OL]. [2014-06-23]. *http://www.cs.umd.edu/~avik/projects/scandroidascaa*.

[59] Lu L, Li Z, Wu Z, *et al*. Chex: statically vetting android apps for component hijacking vulnerabilities [C]//*Proceedings of the* 2012 *ACM Conference on Computer and Communications Security*. New York: ACM, 2012: 229-240.

[60] Wu L, Grace M, Zhou Y, *et al*. The impact of vendor customizations on android security[C]//*Proceedings of the* 2013 *ACM SIGSAC Conference on Computer & Communications Security*. New York: ACM, 2013: 623-634.

[61] Rastogi V, Chen Y, Jiang X. Droidchameleon: evaluating android anti-malware against transformation attacks[C]//*Proceedings of the* 8*th ACM SIGSAC symposium on Information, Computer and Communications Security*. New York: ACM, 2013: 329-334.

[62] Elish K O, Yao D D, Ryder B G, *et al*. A static assurance analysis of android applications [DB/OL]. [2013-06-23]. *http://people.cs.vt.edu/~danfeng/papers/user-intention-PA-2013.pdf*.

[63] Grace M, Zhou Y, Zhang Q, *et al*. Riskranker: scalable and accurate zero-day android malware detection [C]//*Proceedings of the* 10*th International Conference on Mobile Systems, Applications, and Services*. New York: ACM, 2012: 281-294.

[64] Grace M, Zhou Y, Wang Z, *et al*. Systematic detection of capability leaks in stock Android smartphones [DB/OL]. [2014-06-23]. *http://www.cs.fsu.edu/~zwang/files/NDS12_Woodpecker.pdf*.

[65] Gibler C, Crussell J, Erickson J, *et al*.: AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale[C]//*Trust and Trustworthy Computing*. Berlin Heidelberg: Springer-Verlag, 2012: 291-307.

[66] Kim J, Yoon Y, Scandal K Y. SCANDAL: Static analyzer for detecting privacy leaks in android applications[DB/OL]. [2014-06-23]. *http://mostconf.org/2012/papers/26.pdf*.

[67] Chin E, Felt A P, Greenwood K, *et al*. Analyzing inter-application communication in Android[C]//*Proceedings of the* 9*th international Conference on Mobile Systems, Applications, and Services*. New York: ACM, 2011: 239-252.

[68] Dynamic, metamorphic (and opensource) virtual machines[EB/OL]. [2014-06-23]. *http://archive.hack.lu/2010/Desnos_Dynamic_Metamorphic_Virtual_Machines-slides.pdf*.

[69] Staiger S. Static analysis of programs with graphical user interface[C]//*Proceedings of the* 11*th European Conference on Software Maintenance and Reengineering*. Oakland: IEEE, 2007: 252-264.

[70] Christodorescu M, Jha S, Seshia S A, *et al*. Semantics-aware malware detection[C]//*Proceedings of the* 2005 *IEEE Symposium on Security and Privacy*. Oakland:IEEE, 2005: 32-46.

[71] Enck W, Ongtang M, Mcdaniel P D. Understanding Android Security[J]. *IEEE Security & Privacy*, 2009, **7**(1): 50-57.

[72] Zhou Y, Jiang X. Dissecting android malware: Characterization and evolution[C]//*Proceedings of the* 2012 *IEEE Symposium on Security and Privacy*. Oakland: IEEE, 2012: 95-109.

[73] Zhou W, Zhou Y, Jiang X, *et al*. Detecting repackaged smartphone applications in third-party android marketplaces[C]//*Proceedings of the second ACM conference on Data and Application Security and Privacy*. New York: ACM, 2012: 317-326.

[74] Google code. Androguard[EB/OL]. [2014-06-23]. *https://code.google.com/p/androguard/*.

[75] VIAFORESICS. [EB/OL]. [2014-06-23]. *https://santoku-linux.com/*.

[76] Zhao M, Ge F, Zhang T, *et al*. AntiMalDroid: An efficient SVM-based malware detection framework for Android[C]//*2nd International Conference on Information Computing and Applications*. Berlin: Springer, 2011: 158-166.

□