

文章编号:2095-6134(2015)05-0689-06

一种易部署的 Android APP 动态行为监控方法^{*}

王学强^{1,2,3}, 雷灵光^{1,2†}, 王跃武^{1,2}

(1 中国科学院信息工程研究所, 北京 100093; 2 中国科学院数据与通信保护研究教育中心, 北京 100093;
3 中国科学院大学, 北京 100049)
(2014 年 8 月 20 日收稿; 2014 年 11 月 27 日收修改稿)

Wang X Q, Lei L G, Wang Y W. An easy-to-deploy behavior monitoring scheme for Android applications [J].
Journal of University of Chinese Academy of Sciences, 2015, 32(5): 689-694.

摘 要 Android 平台目前已经成为恶意代码攻击的首要目标, 超过 90% 的 Android 恶意代码以 APP 的形式被加载到用户设备. 因此, 监控 APP 行为成为对抗 Android 恶意代码攻击的重要手段. 然而, 已有的监控手段依赖于对 Android 系统底层代码的修改. 由于不同 OEM 厂商对 Android 系统的严重定制, 直接改动商用 Android 系统的底层代码很难由第三方人员部署到用户设备. 本文在分析 Android 进程模型和代码执行特点的基础上, 提出一种在应用层实现的程序行为监控方案, 通过动态劫持 Android 虚拟机解释器的方法, 实现对应用程序代码执行情况的全面监控. 由于不直接对 Android 系统源码进行任何改动, 该方案可以灵活、快速地部署在不同型号、不同版本的 Android 移动终端上. 通过对原型系统的实现和测试, 发现该系统易于部署、监控全面并且性能损耗较低.

关键词 Android APP; 行为监控; Dalvik 劫持; 动态注入
中图分类号: TP309 **文献标志码**: A **doi**: 10. 7523/j. issn. 2095-6134. 2015. 05. 016

An easy-to-deploy behavior monitoring scheme for Android applications

WANG Xueqiang^{1,2,3}, LEI Lingguang^{1,2}, WANG Yuewu^{1,2}

(1 Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China;
2 Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China;
3 University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract Malicious applications pose tremendous threats to Android platform. More than 90% of malicious codes are introduced in the form of Android apps. Hence, behavior monitoring scheme for Android applications are required in order to resolve the problem. However, most of the schemes are based on system customization and hard to deploy on devices for Android's fragmentation problem. In this paper, an easy-to-deploy Android application monitoring method on the basis of process hijacking is proposed after analysis of Android process model and code execution details. The method depends on Dalvik interpreter entry point and system call interception. The authors created a fully usable prototype of the system, and the evaluation results show that the system is easy to

^{*} 国家保密局保密科研项目(BMKY2013B12-2)资助

[†] 通信作者, E-mail: lglei@is. ac. cn

deploy, provides a whole-scale behavior of Android applications, and incurs little performance overhead.

Key words Android APP; behavior monitoring; Dalvik hijacking; dynamic instrumentation

Gartner 统计数据显示,2013 年第 3 季度 Android 的全球智能手机市场占有率为 81.9%,并呈现逐年增长的趋势^[1]. Android 设备的日益增长,吸引了大量的恶意攻击者.根据 Cisco 2014 年度安全报告,2013 年针对 Android 平台的恶意代码数量占移动终端总量的 99%,其中绝大部分恶意代码以 APP 的形式出现^[2]. Android 应用程序行为监控是面对严峻的 Android 安全态势的重要手段.

静态分析. Android 系统中,为了完成特定的功能,应用程序需要申请对应的 Permission. 这些 Permission 申请以明文文本的形式保存在应用程序的 AndroidManifest.xml 文件中. 通过解析这些文本文件即可获得应用程序的 Permission 特征,这些特征在一定程度上反映了应用程序的功能,因此可用于程序行为判断和恶意行为检测. 研究人员据此提出大量基于 Permission 分析的 Android 恶意行为检测方案,如 Enck 等^[3]提出的轻量级检测系统 - Kirin,通过分析应用程序的 Permission 特征,检测应用可能存在的恶意行为. Felt 等^[4]利用 API 和 Permission 的映射关系,分析应用程序使用的 API,发现大约 1/3 应用程序存在过度申请 Permission 的问题. Grace 等^[5]在提取应用程序 Permission 特征的基础上,根据数据流分析结果判断应用程序是否存在 Permission 泄漏问题. 然而,静态分析存在其自身缺陷,例如无法分析混淆或加密过的模块、无法处理程序运行过程中动态加载的代码等. 比如,典型的 root 攻击病毒 DroidKungFu,加密存储用于获取 root 权限的代码,并且只在内存中进行解密和执行^[6],静态分析方法无法检测此类病毒.

动态分析. 动态 APP 行为监控是在 APP 运行过程中,动态地获取 APP 的 API 调用序列等行为信息,并且对其进行干预控制的技术. 很明显,动态分析相对于静态分析,能够更为有效地对 APP 行为进行监控. 根据作用层次的不同,Android 动态行为分析可以分为多种不同的方法. 比如,Blasing 等^[7]引进了沙箱,在沙箱中动态运行应用程序,获取系统调用等底层操作,并根据这些信息

进行恶意代码检测. Burguera 等^[8]通过修改 Android 系统代码,利用 strace 获取应用程序的系统调用信息,并将该信息发送至一个远程服务器,服务器通过对系统调用的聚类分析判断应用程序是否存在恶意行为. Enck 等^[9]通过定制和修改 Android 系统的中间层代码,以添加数据标签和监控数据流向的方式检测应用程序中的隐私泄露威胁. Xu 等^[10]反编译应用安装包,通过程序插桩的方式,监控应用程序的行为,并为其配置安全策略. 沙箱环境(如 Android 模拟器)只能进行 APP 行为分析,难以及时阻断用户设备上发生的恶意行为. 不同 OEM 厂商都对 Android 系统进行了严重的定制,使得只有 OEM 厂商自己才能随意改动其设备上的 Android 系统代码,而第三方难以将改动过的 Android 系统直接部署到设备上. 修改 APP 代码的方式虽然可以直接部署到实际的 Android 设备上,但需要用户确保所有的非可信 APP 均被修改,增加了用户负担. 此外,处理过的 APP 与移动终端上已经安装的同款 APP 采用不同的签名,不能共享应用程序的历史数据,给用户使用带来不便;同时,它无法对预装的系统应用程序行为进行管控.

针对上述问题,本文提出一种能够直接部署在不同 OEM 厂商生产的 Android 设备上的动态 APP 行为监控方法. 不同于已有的方法,该方法根据 Android 系统基于 Linux 的特点,采用动态注入的手段,实现对 APP 进程空间的动态修改. 通过这种动态注入,可以在 APP 运行时将监控代码植入 APP 进程空间的 Dalvik 虚拟机代码中,实现 APP 行为监控. 相对于修改 APP 代码的方法,该系统在确保易部署性的条件下,使得监控代码对 APP 透明,APP 代码可以在不做任何修改的情况下安装到用户设备,实现其行为监控. 相对于直接修改 Android 系统代码的方法,该方法不需要对 Android 系统做任何修改,保证监控代码可以容易地部署在实际 Android 设备上. 该方法直接作用于被监控 APP 的 Dalvik 执行环境,在获取 APP 动态行为信息的同时又能够有效阻断恶意行为的发生.

1 背景知识

1.1 应用程序代码架构

Android 基于 Linux 内核,采用支持 Java 和本地(C/C++)代码的应用程序架构,如图 1 所示.应用程序的 Java 代码建立在 Framework 层 API 的基础上,其字节码由 Dalvik 虚拟机读取并解释执行.同时,应用程序或 Android 系统的 Java 代码可通过 JNI(Java Native Interface)等方式与本地代码进行交互. Android NDK 支持在本地代码中动态加载 Java 类实例,并调用 Java 方法.这种从本地代码调用的 Java 方法同样会经过 Dalvik 解释器的解析过程.由于 Android 应用程序中发送短信、读取通讯录和访问位置信息等功能均需要调用 Framework 层的 Java 接口,因而可在应用程序进程空间的 Dalvik 虚拟机代码中监控该应用的行为.

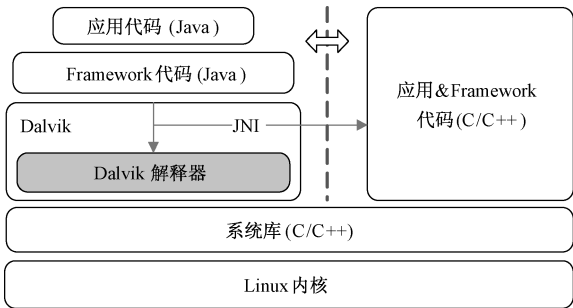


图 1 Android 应用程序代码架构图

Fig. 1 Architecture of Android APP code

1.2 Android 进程模型

不同于 Linux 的进程创建方式,Android 应用程序进程由 zygote 负责创建. zygote 进程加载了大量系统库和 Android 系统资源包,并维护 Dalvik 虚拟机实例,该实例负责执行 dex 字节码,并完成堆栈管理、线程管理、垃圾回收、对象管理等重要功能. 用户点击桌面上的应用图标后,Android 的“activity”服务会判断该应用是否已经启动. 若否,“activity”服务通过 zygote socket 将应用创建请求发送给 zygote 进程. zygote 创建子进程,并依据请求消息中的 UID、GID 等配置子进程的状态. 这样,新创建的子进程继承了 zygote 加载的系统资源等运行环境,拥有执行 dex 字节码的能力. 因而,通过监控 zygote 进程的 fork 系统调用,可以及时地发起对应用程序的管控.

2 系统设计

2.1 设计思路

本工作的目标是监控 Android APP 的行为. 由于动态监控在处理混淆或动态加载的代码模块上存在自身优势,并且,如 1.1 所述,APP 的行为可以从 APP 进程的 Dalvik 运行时环境中获取,因而,本工作采用动态 Dalvik 监控的方法. Dalvik 监控可以通过多种方式实现,如直接定制 Android 系统的 Dalvik 代码或者对 APP 代码进行静态插桩,但是经分析,这些方式在可部署性上存在问题. 我们发现,动态地修改 APP 进程加载的 Dalvik 系统库可以完成动态监控的工作,并且,无需直接修改系统和 APP 的代码,可以保持系统的易部署性. 因此,本工作通过将监控代码动态注入 APP 进程的 Dalvik 模块的方式,获取 APP 调用 Java 方法(直接调用或反射)的行为序列,并达到控制程序执行流程的目的.

另一方面,为了监控到 APP 的完整生存周期,需要在 APP 进程启动之初完成监控代码动态注入的工作. 如 1.2 所述,Android 设备上安装的每个 APP 都被分配了唯一的 UID,并且 APP 进程由 zygote 进程 fork 而来,所以监控 zygote 创建进程的行为并匹配该进程的 UID,就可以探测到 APP 进程的启动,并及时地将监控代码注入到该进程的地址空间,从而监控到 APP 整个生存周期的行为. 系统架构围绕上述 2 点设计,保证完成 APP 监控的功能性和完整性.

2.2 系统架构

本文提出的应用程序行为监控系统的架构如图 2 所示,包含 5 个模块.

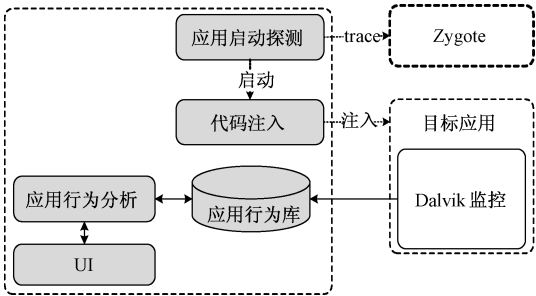


图 2 应用程序监控原型系统架构图

Fig. 2 Architecture of monitoring system

- UI:提供相关的用户操作界面,供用户选择待监控的应用程序. 同时,在应用程序执行过程

中,实时获取敏感操作并呈现给用户.

- 应用启动探测:通过监控 `zygote` 进程,判断目标应用是否正在启动,若是,则立即触发监控.应用启动探测模块保证行为监控覆盖应用程序的整个生命周期.

- 代码注入:负责监听应用启动探测模块的请求,将监控代码注入到被监控应用的进程空间中.

- Dalvik 监控:该模块被注入到目标应用程序的进程空间中,劫持解释器的关键函数,解析 Java 方法的内容,然后通过进程间通信反馈行为日志至应用行为库.

- 应用行为库:存放从监控模块获取的目标应用程序的调用信息,实时提供给行为分析模块.

- 应用行为分析:结合用户定义的敏感操作,采用字符串匹配、行为序列分析等手段分析目标应用程序行为,检测其执行过程中可能存在的恶意行为.

3 系统实现

本文所述行为监控系统的核心是基于代码注入的 Dalvik 监控模块和基于系统调用监控的应用启动探测模块,本节对二者的实现原理和关键细节进行介绍.

3.1 Dalvik 监控

Dalvik 的运行时数据区结构如图 3 所示.方法区和 Dalvik 堆存放方法和对象的定义,为各线程共享.解释栈在线程创建时分配,与线程拥有相同的生命周期,Java 方法的执行均需要经过解释栈入栈和出栈操作.

图 4 显示 Dalvik 虚拟机内部的细节. Dalvik 虚拟机读取方法帧后,判断该方法帧是否为 JNI 方法.若是,根据 `DalvikBridgeFunc` 的定义跳转到本地函数执行.若否,会判断当前解释器运行模式,如果处于 `int:fast` 高速模式或 `int:jit` 即时编译模式,则解释器入口为 `dvmMterpStdRun`,该函数由汇编代码实现,执行效率较高;如果解释器处于 `int:portable` 可移植模式,则入口为 `dvmInterpretStd`,该函数由 C/C++ 实现,具有较好的可移植性. Java 方法监控的实现可概括如下:采用 `inline hook` 技术,根据解释器工作模式,劫持相应的入口函数和解释器执行路径上的关键函数,分析解释器栈状态和当前执行的 Java 方法.

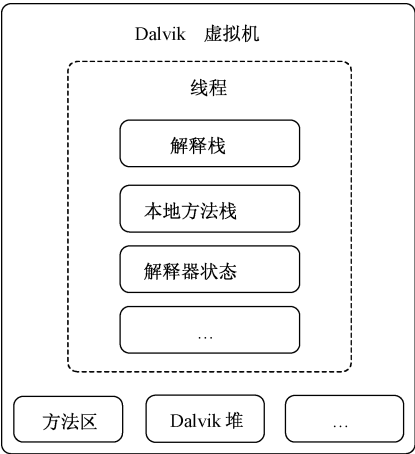


图 3 Dalvik 运行时数据区
Fig. 3 Runtime data areas of dalvik

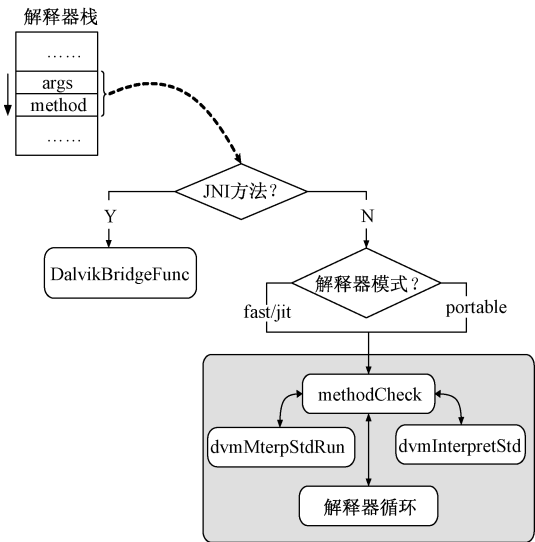


图 4 Dalvik 虚拟机内部细节
Fig. 4 Inside of dalvik virtual machine

Java 方法解析的工作集中在参数解析, Dalvik 虚拟机内部为每个方法维护一个 `Method` 结构,该结构指定方法名、类名、参数类型等信息.特别地,引用类型的参数,指向 Dalvik 堆上的对象,利用 JNI 字段描述符可知对象的具体类型(如 `Landroid/content/Intent;` 表示对象为 `Intent`). 本系统通过枚举 `ClassObject` 的数据域定义,将复杂的引用对象规约为基本类型或 `String` 等简单引用类型,分析复杂的 Java 对象.比如从 `Intent` 对象的 `Action`、`Url` 等数据域,可以获得读取联系人、发送短信、上网等行为信息.表 1 显示了监控到的一条 Java 方法调用,其中的 `sms:10010` 表示发送短信到 10010 的行为.

表 1 Java 方法描述

Table 1 Description of Java method

属性名	属性值
ProcessName	com. example. android. Msg,
ThreadName	Main
Method	Landroid/app/Instrumentation; . checkStartActivityResult(IL)
Argument	Intent[mAction:[android. intent. action. SEND] ,Uri:[smsto:10010]]

不同 Android 系统版本的虚拟机实现不尽相同. 如 Android 2.3 解释器入口函数 dvmMterpStdRun 的参数为 InterpState 指针,而 Android 4.0 及更高版本采用的是 Thread 指针参数. 因而,本系统根据 Android 版本的不同,分别定制 Java 方法监控模块.

3.2 应用启动探测

应用启动探测模块负责实时监控应用程序的启动,及时对目标应用发起进程注入,使行为监控覆盖到应用程序的整个生命周期. 根据 1.2 节所述,应用程序由 zygote 进程 fork 而来,通过监控 zygote 进程的 fork 系统调用可以实时探测到应用程序启动. 通过调研 strace、ptrace、kprobes 等几种系统调用监控方式,发现 ptrace 无需修改系统代码,能够对系统调用实现灵活的监控. 而 strace、fstrace、kprobes 或者管控能力较差,或者依赖于 Linux 内核的某些特性(如 kprobes 需要内核开启 LKM 支持),无法灵活使用.

在本方案中,通过设置 ptrace 的 PTRACE_EVENT_FORK 等选项,监控 zygote 进程创建子进程的所有操作. 因为每个应用程序拥有一个唯一的 UID,所以可利用 UID 将新进程映射到应用程序. 这样,保证在第 1 时刻探测到应用程序的启动并发起监控,使监控覆盖到目标应用程序的完整生命周期.

4 实验

本节从监控应用程序行为的性能以及本系统带来的损耗 2 个方面进行实验分析.

4.1 性能测试

为测试本系统性能,我们获得了豌豆荚上下载量排行前 1 050 的应用,并利用该系统动态监控应用的发送短信行为. 首先通过静态分析过滤出其中存在短信发送能力的 113 个应用(即申请

了 SEND_SMS 权限的应用);然后利用事件流生成工具 monkey 触发执行筛选出的 113 个应用. 在整个监控过程中发现,35 个应用程序执行了发送短信操作,涉及 40 个手机号码,其中包括 1 065 * 等可能造成资费消耗的定制服务号码.

4.2 性能损耗

本节分别从应用程序启动效率和运行效率 2 个方面评估性能损耗.

4.2.1 启动效率

应用启动探测模块基于监控 zygote 的系统调用,而应用程序进程由 zygote 创建,所以本系统影响了应用程序启动效率. 本实验,通过 startService 启动一个远程服务进程,计算发出 startService 指令和执行服务的 onCreate 方法的时间差. 该过程在正常模式和监控 zygote 进程的情况下各测量 10 次,测试了 Nexus S(Samsung)、Meizu MX II、Sony LT29i 和 Samsung Galaxy Note II 这 4 款手机,发现整个过程的性能损耗维持在 23 ms 之内,因为新创建应用进程的行为出现频率较低,所以本系统不会明显影响用户体验. 结果如图 5.

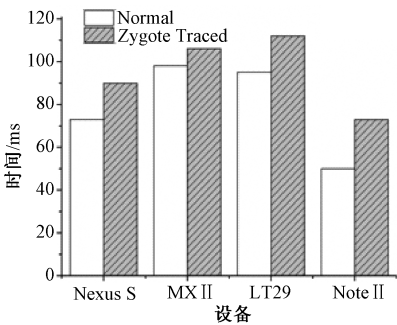


图 5 应用程序启动效率

Fig. 5 APP launching efficiency

4.2.2 运行效率

本系统通过劫持 Dalvik 虚拟机解释器的关键函数进行应用程序行为监控,因此系统执行会对 Dalvik 虚拟机的执行效率产生影响. 在实验中选用搭载 Android 4.1.2 系统的 Nexus S 测试手机,在“正常情况”和“启动监控”2 种情况下,通过 CaffeineMark 3.0 benchmark,测试本系统对虚拟机多项指标的影响. 测试结果如图 6,行为监控引起的整体性能损耗较小,约为 8.5%,由于在行为监控中主要对 method 和 string 进行分析,所以这 2 项指标的性能损耗较大,分别为 16% 和 11%,而 sieve、logic、loop 和 float 4 项指标的损耗

均低于 7%.

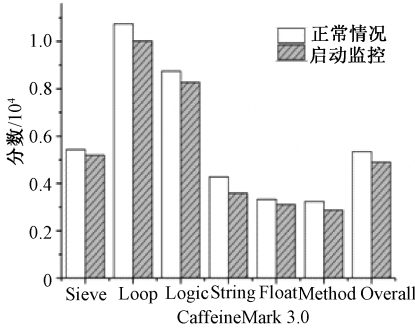


图 6 Dalvik 虚拟机监控的性能损耗

Fig. 6 Overhead of Dalvik virtual machine

5 结语

本文通过对 Android 进程模型、应用程序代码架构和系统机制的分析与研究,利用进程劫持的方法,提出一种易部署的 Android 应用程序行为监控方法,以分析应用程序的行为;同时基于该行为监控方法,设计出一套应用程序行为监控系统. 通过原型系统的实现和多次实验分析,表明本文提出的行为监控方法以及相应的应用程序行为监控系统具有易部署、性能损耗小的特点,可以满足 Android 移动设备安全监控的需求.

在谷歌 I/O 2014 开发者大会上,Android L 正式发布,Android Runtime (ART) 取代 Dalvik 成为默认运行时,ART 支持将应用程序的 Java 代码编译成 .oat 格式. 作为 Dalvik 层监控的扩展,可以在采用 ART 的移动终端上对 .oat 文件进行静态插桩,完成对应用程序行为的监控.

参考文献

[1] Gartner. Gartner says smartphone sales accounted for 55 percent of overall mobile phone sales in third quarter of 2013 [EB/OL]. (2013-11-14) [2014-07-20]. <http://www.gartner.com/newsroom/id/2623415>.

[2] Cisco. Cisco 2014 annual security report [R/OL]. Cisco_2014_ASR. pdf. (2014) [2014-07-20]. https://www.cisco.com/web/offers/gist_ty2_asset/

[3] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification [C] // Proceedings of the 2009 ACM Conference on Computer and Communication Security (CCS). 2009; 235-245.

[4] Felt A P, Chin E, Hanna S, et al. Android permissions demystified [C] // Proceedings of the 2011 ACM Conference on Computer and Communication Security (CCS). 2011; 627-638.

[5] Grace M C, Zhou Y, Wang Z, et al. Systematic detection of capability leaks in stock android smartphones [C] // 19th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, 2012.

[6] Jiang X X. Security alert: new sophisticated Android malware DroidKungFu found in alternative Chinese app markets [EB/OL]. (2011-06-23) [2014-07-20]. <http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html>.

[7] Bläsing T, Batyuk L, Schmidt A D, et al. An android application sandbox system for suspicious software detection [C] // 5th International Conference on Malicious and Unwanted Software (MALWARE). 2010; 55-62.

[8] Burguera I, Zurutuza U, Tehrani S N. Crowdroid: behavior-based malware detection system for android [C] // Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM). ACM, 2011; 15-26.

[9] Enck W, Gilbert P, Chun B G, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones [C] // Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2010; 393-407.

[10] Xu R, Saïdi H, Anderson R. Aurasium: Practical policy enforcement for android applications [C] // Proceedings of the 21st USENIX Conference on Security Symposium. USENIX Association, 2012; 539-552.