一种 Android 应用程序恶意行为的静态检测方法①

李子锋, 程绍银, 蒋 凡

(中国科学技术大学 信息安全测评中心, 合肥 230027)

摘 要: 目前 Android 应用程序的安全问题得到越来越多的关注. 提出一种检测 Android 应用程序中恶意行为的静态分析方法, 该方法采用静态数据流分析技术, 并实现了常量分析算法, 通过跟踪应用程序对常量值的使用来检测恶意订购、资费消耗等多种类型的恶意行为. 实验结果表明, 该方法可以有效检测出 Android 应用程序的恶意行为, 具有较高的实用性.

关键词: 静态分析; 恶意行为; 常量分析; Android 安全; 数据流分析

Static Detection Method for Malicious Behavior in Android Apps

LI Zi-Feng, CHENG Shao-Yin, JIANG Fan

(Information Technology Security Evaluation Center, University of Science and Technology of China, Hefei 230027, China)

Abstract: Currently, the issues on Android application's security have attracted more and more attentions. This paper presents a static analysis method to detect malicious behavior in Android applications. The method uses static data flow analysis technology, and implements a const analysis algorithm that tracing how the const value is used by the application to detect different kinds of malicious behavior, such as ordering services and consuming payments maliciously. The result of experiments shows that the method is practical, and can detect the malicious behavior in Android applications effectively.

Key words: static analysis; malicious behavior; const analysis; Android security; data flow analysis

1 引言

目前 Android 系统的移动设备已经得到了广泛的应用. 2012 年上半年搭载 Android 系统的智能终端出货量达到一亿多部, 市场占有率近 70%[1]. 随着移动智能终端的发展, Android 平台上的安全问题也日益突出, 恶意软件的数量出现爆发式的增长. 仅 2012 年第三季度就查杀恶意软件 2.3 万余款, 感染手机 991 万部, 且 94%的恶意软件集中在 Android 平台[2]. 而目前主流的安全软件都无法有效遏制恶意软件的大范围传播, 这使得 Android 用户面临着越来越大的安全威胁.

目前对 Android 恶意应用程序的检测方法主要有两种: 一种是按照病毒查杀的方式进行检测, 将恶意软件当成病毒, 按照特征签名的方式进行检测. DroidRanger[3]总结了十种已知的 Android 恶意软件的行为特征和两种启发式规则, 用来检测未知应用程序. 该方法可以快速检测出已知恶意软件, 对于新出现的应用程序多采用人工分析的方式, 因此检测结果存在一定的滞后期. 另一种方式是采用动态监控的方法, 通过动态实时监控应用程序的执行及其与外部环境的交互进行检测. TaintDroid[4]是一个系统级的动态实时分析工具, 采用动态数据流分析方法, 监控应用程序对敏感信息的泄露. AppInspector[5]采用动态分析方法, 可以自动生成输入并在程序执行过程中记录日志, 通过分析记录的日志信息来检测应用程序当中是否存在隐私泄露行为. 该方法依赖于特定的触发条件, 对于触发逻辑较为复杂的应用程序, 存在明显的不足.

本文提出了一种基于常量分析的静态数据流分析方法, 通过静态分析 Android 字节码文件, 跟踪应用程

① 基金项目:高等学校博士学科点专项科研基金新教师类资助课题(20113402120026);安徽省自然科学基金(1208085QF112);
安徽省高等学校优秀青年人才基金(2012SQRL001ZD);中央高校基本科研业务费专项资金(WK0110000007)
收稿时间:2012-12-27;收到修改稿时间:2013-01-29

序对常量值的使用, 检测应用程序中存在的恶意订购、资费消耗、执行系统命令、加载本地代码等多种恶意行为. 文中对该方法进行了详细介绍, 大规模的实验表明, 该方法可以有效检测出应用程序当中存在的恶意行为, 具有较高的实用价值.

2 方法概述

方法总体架构图如图 1 所示. 输入是 Dalvik 字节码文件, 在分析之前, 首先对 Android 应用程序(后缀为.apk)解包, 提取出 Dalvik 字节码文件(后缀为.dex).

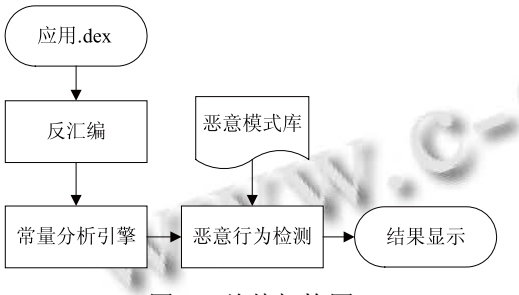

图 1 总体架构图

检测 Android 应用程序恶意行为的过程包括如下四个步骤:

(1) 反汇编模块对输入的字节码文件进行反汇编, 收集需要的程序信息, 如指令结构、基本块结构、函数结构、类结构、函数调用图、控制流图等程序结构, 构建字符串表保存程序中出现的固定字符串信息.

(2) 根据反汇编模块保存的反汇编信息, 构建常量分析引擎, 模拟应用程序的执行过程, 跟踪常量值在应用程序中的传播过程, 保存关键函数调用点的程序状态.

(3) 根据常量分析引擎保存的程序状态, 结合恶意模式库中的检测规则, 进行恶意行为检测, 根据检测到的恶意行为按照危险类别进行分类保存.

(4) 根据检测结果生成恶意行为的详细说明, 详细描述当前恶意行为的触发流程、关键参数的常量值及带来的危害等信息. 提供图形化显示, 方便查看.

3 关键模块的设计与实现

3.1 常量分析引擎

常量传播是在模拟程序执行的基础上, 跟踪程序中的常量值(固定字符串、立即数等)的使用情况. 常量值在应用程序中扮演着很重要的角色, 如手机号码、短信内容、URL 等都可以是一个固定字符串. 有些应用通过使用这些常量值来产生恶意行为, 如将固定内容的短信发送到一个固定的字符串号码订购 SP 服务. 常量分析引擎通过跟踪应用程序中常量值的使用情况, 在危险函数调用点进行常量值检测, 如果关键参数的常量值满足恶意模式库中定义的规则, 则说明检测到了恶意行为.

常量传播是一种基于静态数据流的分析方法. 静态数据流分析是一种在安全领域广泛使用的技术[6], 一般包括过程内分析和过程间分析两个阶段: 过程内分析阶段求解当前函数单元中指令的执行顺序, 根据当前函数的控制流图, 采用图的遍历算法, 以基本块为单位, 保证每个基本块至少被执行一次; 过程间分析阶段根据函数调用点的信息, 在函数调用图上求解实际应该被调用的函数, 根据函数调用点的参数类型, 计算实际应该被调用的函数, 并模拟函数调用的过程, 将程序流程转到被调函数单元. 对于过程内分析和过程间分析的研究较多, 见参考文献[7].

3.1.1 隐式函数调用

Android 系统中存在大量的隐式函数调用, 如事件触发、多线程、接口函数、反射等. 如果不能正确找到被调函数, 会大大降低静态分析的精度. 因此, 需要在过程间分析阶段对隐式函数调用进行处理, 根据函数调用点的上下文信息动态查找实际执行的函数.

隐式函数调用涉及以下四个方面:

(1) 接口函数调用. Dalvik 字节码对接口函数调用采用 invoke-interface 操作码, 据此可以识别出所有的接口调用. 接着在接口函数调用点找到实际被调用的函数来执行. 由于在过程间分析中我们维护了类型信息, 因此可以准确找到调用函数所在对象的类名. 接下来只要从该对象中找到相对应的函数调用即可. 需要注意的是, 此时的类名可能有多个, 对每种可能, 需要依次调用接口的实现函数.

(2) 多线程机制. 线程的生命周期是应用程序在运行时系统进行动态维护, 如用户调用线程的 start 方法时, 系统会自动调用 run 方法, 由于没有显式的调用, 在静态分析时, 需要静态模拟线程的生命周期. Java 有两种实现多线程方法, 一种方法是继承 Thread 类, 通过运行 Thread.start 接口来启动, 可以按照接口问题的解决方法, 寻找实际调用对象的类型, 不过这需要一个额外的步骤, 即根据 start 函数去实际类中寻找 run

函数. 另一种方法是实现 Runnable 接口, 然后直接根据子类名调用 start 函数, 省去了查找实际类型的步骤, 直接查找 run 函数即可.

(3) 事件触发机制. 这在 Android 应用程序中使用非常广泛. 如对某个按钮注册响应函数 onClick. 由于无法准确知道响应函数在何时被调用, 因此在响应函数注册时就调用响应函数进行分析.

(4) 反射. 反射是一种可以根据函数名(字符串形式)来调用实际的函数的一种特殊机制. 首先通过 Class.forName 函数找到实际的类, 然后通过函数名动态查找实际的函数进行调用. 为了准确对反射进行模拟, 不仅需要类型信息来识别反射, 还需要类名和函数名的字符串信息来找到实际的函数. 我们实现了常量分析模块来维护字符串在程序中的传播过程, 因此当遇到反射时, 通过查找我们维护的类结构可以很容易找到实际调用的函数.

3.1.2 常量传播

常量传播过程分为两个阶段: 常量值的引入和常量值的传播.

常量值的引入. 常量值一般有两种, 一种是立即数, 一种是固定字符串. 这些常量值一般是通过 const 指令引入到程序当中. 例如对于指令 *const-string* *v*1, "*abc*", 该指令定义了一个变量 *v*1, 且 *v*1 的值为固定字符串"abc". 为了跟踪常量值在应用程序中的使用, 设置了专门的常量值类型来保存程序中的常量值. 如对于上面的指令, *v*1 将被标记为常量状态, 并与常量值"*abc*"进行关联.

表 1 常量传播表(*v*1 依赖于 *v*2 和 *v*3)

v1	*v2*	*v3*
Const	Const	Const
CDV	Const	Non-const
CDV	Const	CDV
CDV	Non-const	CDV
Non-const	Non-const	Non-const
CDV	CDV	CDV

常量值的传播. 常量值的传播根据指令的语义对常量值进行传递. 例如, 指令 *move* *v*1,*v*2 的含义是将 *v*2 寄存器的值赋给 *v*1. 如果 *v*2 为常量, 则将 *v*1 也置为常量状态. 对于数值运算指令, 如 *add* *v*1,*v*2,*v*3, 该指令的含义是 *v*1=*v*2+*v*3. 如果 *v*2 为常量值, *v*3 为非常量值, *v*1 是否该标记为常量值? 为解决此问题, 引入了一个称为常量依赖值(Const Dependence Value, CDV)的类型. *v*2 为常量值, *v*3 为非常量值, 则 *v*1 为常量依赖值, 且 *v*1 依赖于常量 *v*3. 表 1 为常量传播表, 其中 *v*2、*v*3 为源变量, *v*1 为目的变量.

3.2 恶意行为检测

一般而言, 应用程序中的恶意行为都是通过调用系统库函数实现的. 可以通过检测一些危险函数调用点处的参数信息, 来判断是否存在恶意行为. 为了识别尽可能多的恶意行为, 本文对 Android 开发文档[8]进行收集整理, 并结合一些已经的恶意行为模式, 构建了一个恶意模式库, 库中根据潜在危险函数参数信息的不同, 标记为不同类型的危险行为. 比如对于发送短信的库函数 SmsManager.sendTextMessage(…), 第 1 个参数为目的号码, 如果目的号码为一个 SP 号码, 则可能是一种订购 SP 服务的行为.

恶意行为检测算法分为以下四个步骤:

(1) 敏感行为识别. 在系统函数调用点处识别可能存在恶意行为的敏感函数. 根据恶意模式库中定义的危险函数名和参数信息, 与被调函数进行匹配, 如果匹配成功, 则识别出一条敏感行为.

(2) 危险行为匹配. 对步骤(1)识别出的敏感行为进行危险确认, 通过详细分析当前函数调用点处的参数个数、类型和常量值信息, 并根据检测规则库中的详细配置信息进行危险行为识别.

(3) 危险等级确定. 对步骤(2)匹配成功的危险行为确定其危险等级. 先根据当前恶意行为对应的函数名制定一个初步的危险等级, 再根据关键参数的值信息对当前危险等级进一步调整, 以确定最终的危险等级.

(4) 检测结果保存. 保存检测出的恶意行为的详细信息, 给后面的结果显示提供丰富的输入.

4 实验评估

根据上述方法实现了一个原型系统, 反汇编模块主要借助商业化的反汇编工具 IDA pro 实现反汇编信息的收集, 后面的分析检测等模块用 Java 语言实现, 可以检测恶意订购、恶意传播、资费消耗、本地代码、广告链接等五种类型的恶意行为, 恶意行为的分类和描述如表 2 所示. 为了更好的描述不同恶意行为的危害程度, 将不同类型的恶意行为分为高中低三个等级.

为了验证方法的有效性, 对某安全公司提供的 1767 个 Android 恶意软件进行实验. 实验结果如图 2 所示. 检测到 1702 个(96.3%)应用当中存在不同等级的恶意行为. 其中检测到含有高危险行为的应用程序 631 个(35.7%), 不含高危险而含有中危险行为的应用

有 875 个(49.5%), 不含高危险和中危险只含有低危险行为的应用有 195 个(11.0%). 由于只关心以上五种类型的恶意行为, 且受到静态方法的限制, 分析过程中存在一定程度的不准确性, 有 65 个应用(3.7%)未检测到任何恶意行为.

表 2 恶意行为分类

恶意类型	描述	等级
恶意订购	主要包括订购 sp 服务的行为	高
恶意传播	自动下载 APK，群发短信等	高
资费消耗	向固定的号码或未知号码发送短信	中
本地代码	执行系统命令，加载本地库（.so）文件	低
广告链接	固定的广告网址	低

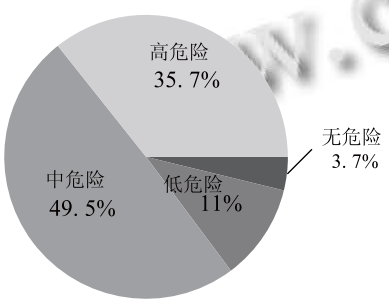

图 2 危险等级分布

表 3 检测出的恶意行为分布

恶意类别	危险行为数	APK 数	占恶意应用百分比
恶意订购	179	77	4.5%
恶意传播	214	176	10.3%
资费消耗	5758	1167	68.6%
本地代码	4654	679	40.0%
广告链接	9918	968	56.9%

对存在恶意行为的应用按照恶意行为的类型进行统计, 如表 2 所示. 由检测结果可知, 4.5%的应用当中存在恶意订购的行为, 通过直接向 sp 服务商发送短信订购付费服务. 10.3%的应用当中存在恶意传播的行为, 68.6%的应用中存在资费消耗的行为, 该行为通过调用发送短信的函数实现, 短信号码未知. 40%的应用存在执行本地命令或加载本地库文件的行为, 执行系统的命令或加载的本地代码中均有可能存在提权行为, 由于本系统无法分析这些本地库文件, 所以都标记为低危险的行为, 希望能引起警示. 56.9%的应用当中存在着广告链接行为, 目前大多数免费应用当中都存在广告链接的行为, 且广告一般都以图片的形式展示, 是一种消耗用户流量的行为, 标记为低危险级别.

此外, 还发现了使用非常广泛的酷 6 播放器(v2.5.8)中存在恶意扣费的行为, 向 1065502180988 和 10690882 两个 SP 服务商发送短信. 进一步的手工分析验证了确实存在恶意行为. 也确实有用户在使用过程中发现并投诉了该恶意行为.

5 结语

本文提出一种检测 Android 应用程序恶意行为的方法, 该方法采用静态数据流分析技术, 模拟应用程序的执行过程, 通过跟踪应用程序对常量值的使用, 可以检测恶意订购等多种类型的恶意行为. 通过对 1767 个恶意样本进行实验, 验证了方法的有效性. 考虑到目前越来越多的应用程序当中存在本地代码调用, 对本地代码进行安全检测是下一步研究的重点.

参考文献

1 移动终端白皮书(2012 年).http://www.cttl.cn/txyy/ggl/201204/P020120413505417116578.pdf.

2 2012 年第三季度全球手机安全报告.http://cn.nq.com/neirong/2012Q3.pdf.

3 Zhou YJ, Wang Z, Zhou W, Jiang XX. Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. Proc. of the 19th Network and Distributed System Security Symposium(NDSS 2012). San Diego, CA, February 2012.

4 Enck W, Gilbert P, Chun B, Cox LP, Jung J, McDaniel P, Sheth AN. Taint Droid: an information-flow tracking system for realtime privacy monitoring on smartphones Proc. of the 9th USENIX. Vancouver, BC, Canada. 2010: 1–6.

5 Gilbert P, Chun BG, Cox LP, Jung J. Vision: Automated Security Validation of Mobile Apps at App Markets. Proc. of the International Workshop on Mobile Cloud Computing and Services. USA: ACM. 2011: 21–26.

6 Chess B, McGraw G. Static analysis for security. IEEE Security and Privacy, 2004, 2(6): 76–79.

7 Cheng S, Yang J, Wang J, Wang J, Jiang F. Loongchecker: Practical summary-based semi-simulation to detect vulnerability in binary code. Proc. 10th Int. Conf. on Trust Security and Privacy in Computing and Communications. IEEE, 2011: 150–159.

8 Android SDK.http://developer.android.com/sdk/index.html