# 安卓应用程序误用用户信息的监测与控制

廖明华

(华南师范大学增城学院 计算机系, 广州 511363)

**摘 要**: 针对 Android 应用程序常被不完全地审查, 不充分的隔离, 且毫无限制地被用户安装所引起的用户私有敏感信息的泄露, 通过采用动态污点分析技术监测敏感信息何时通过不可信的应用程序离开系统, 同时根据需要采用无害的影子数据遮蔽敏感信息, 或者阻断泄露私有信息的通信, 防止应用程序访问用户希望保密的数据, 在系统的层面上实现实时监测和控制 Android 应用程序使用用户私有敏感信息.

**关键词**: 安卓; 用户私有敏感信息; 动态污点分析; 信息流追踪; 隐私控制

## Monitoring and Control for Android Application in Misusing Users' Private Information

LIAO Ming-Hua

(Department of Computer Science, Zengcheng College of South China Normal University, Guangzhou 511363, China)

**Abstract:** Smartphone applications are frequently incompletely vetted, poorly isolated and installed by users without restraint. Such behavior causes users' private sensitive information leakage. By means of employing dynamic taint analysis technology, our work detects when sensitive data leaves the system via untrusted applications. Meanwhile, according to the need, it uses harmless data to cover the sensitive information, or cuts off the exfiltration communication. Then it can prevent an application to access the data which user wants to keep confidential, achieve real-time monitoring and control for Android applications in misusing the user private information from system level.

**Keywords:** Android; users' private sensitive information; dynamic taint analysis; information-flow tracking system; privacy control system

Android 智能手机操作系统凭借开放性、丰富的硬件选择、不受任何限制的开发商、无缝结合的 Google 应用等优势, 迅速成为最流行的主流手机平台之一. 然而, Android 应用程序常被不完全地审查, 不充分的隔离, 且毫无限制地被用户安装. 这种行为状态包藏了危险: 隐含恶意逻辑或致命缺陷的应用程序有可能窃取用户的私有敏感信息, 回传给网络用于广告和分析, 甚至追踪用户的活动. 很不幸, 应用程序市场在验证应用程序安全方面是个很不完善的代理. 针对以上情况, 本文全面地研究 Android 应用程序误用用户私有信息的情况, 并提出了相关的解决方案.

## 1 Android隐私风险

文献[1]通过抽样方法, 从 2010 年 11 月电子市场上市的 22 个种类的应用程序中, 选取了每一个种类中的最流行的 50 个应用程序. 然后, 通过采用 Android 的 aapt 工具解析应用程序的 Manifest 文件, 测试这组包含 1100 个应用程序的实例. 实验结果显示有 11 个权限引起了 12 种私有敏感信息的暴露: 位置信息, 手机状态(手机号码、IMEI(国际移动电话设备识别码)、IMSI(国际移动用户识别码)、ICC-ID(SIM 卡的序列号)、通话状态), 通讯录, 用户账户信息, 相机, 麦克风, 浏览器历史记录和标签, 日志, SMS 信息, 日历和订阅信息; 同时发现 605 个应用程序(55%)请求访问其中的至少一种资源和因特网, 进而导致不必要的泄露.

## 2 动态污点分析框架

基于动态污点分析的技术已经成功地应用于应用程序的安全性领域, 同时研究人员也在探索该技术在

收稿时间:2013-10-24;收到修改稿时间:2013-11-22

其它不同领域的应用，如程序理解(program understanding), 软件测试和调试[2,3]等.

简单地说, 动态污点分析 (又称为动态信息流分析)是一种在程序运行时, 标记、跟踪和分析特定数据的实时检测技术. 在应用程序的安全性方面，动态污点分析的方法已成功地用于阻止范围广泛的攻击，包括缓冲区溢出攻击[4,5]，格式字符串攻击[5,6]，SQL 和命令注入攻击(command injections)[7]，和跨站点脚本攻击(cross-site scripting)[8].

动态污点分析的总体框架可划分为三个模块: 污染源(taint sources)，传播策略(propagation policy)，和污点池(taint sinks). 也可以划分成三个阶段污点数据标记，污点数据跟踪，污点数据误用检测[9]. 通过更改这三个模块的不同内容，我们可以定义不同的动态污点分析方法.

### 2.1 污点源

污染源是指程序数据(内存地址)被污点标记初始化的描述，在这里指定用什么样的污点标记来标记或污染特定的敏感数据. 程序数据包含了许多不同的类型，包括变量名(variable names)，函数返回值(function-return values,)，数据读取和 I/O 流，如文件或网络连接.

### 2.2 传播策略

传播策略描述了在执行时污点标记怎样传播. 污点传播可以概括为以下步骤: 给定一个语句 s，我们把由 s 所产生的数据称为起始数据(produced data)，控制污点标记分配的函数称为映射函数(mapping function)，影响 s 结果的数据称为影响数据(affecting data). 识别起始数据是容易的，他是数据的集合，存储在寄存器或内存中，它的值会随着 s 的执行而发生改变. 相反，有不同的方式识别影响数据和定义映射函数. 由于我们的目标是提供足够的灵活性，所以在我们的框架中，我们让用户自己定义传播策略的识别影响数据，定义映射函数.

### 2.3 污点池

简单的说，污点池是代码中的一个地址，在这里用户想要对一个或多个存储单元的污点标记进行检查. 污点池可从四个方面分析: 1) ID; 2) 内存地址(memory location); 3)代码地址(code location)和 4) 在某一代码地址结合关联该内存地址的污点标记执行一个或多个检查操作. ID 是一个被用户分配给一个汇点(sink)或一组汇点的整数值.

### 2.4 简单实例

举个简单的例子，用来说明污点源，传播策略，和污点池怎样共同工作，如图 1(a)中的代码. 假设我们的框架的用户对检测参数 a 的值是否影响 y 的值感兴趣, y 值在函数 foo 的最后打印输出. 这种情况下, 用户指定 foo 的第一个参数为污点源, 且指明它被给定的污点标记 ta 进行标记. 然后, 用户将选择默认的基于数据和基于控制流的传播策略. 最后, 用户指定包含变量 y 的 line11 为污点池, 同时它的相关程序操作作为 line11 中 y 的输入污点信息, 然后检查汇点是否包含污点标记 ta(或简单地在随后的检查中记录污点信息).

```
1 void foo (int a) {
2    int x, y;
3    if (a>10) {
4    x=1;
5    }
6    else {
7    x=2;
8    }
9    y=10;
10    print(x);
11    print(y);
12   }
```
(a)

```
1 void foo (int a) {
2    int x, y;
3    x=2;
4    if (a>10) {
5    x=1;
7    }
8    y=10;
9 prints(x);
10    }
```
(b)

图 1　包含隐性信息流的实例代码

## 3　Android平台上私有敏感信息的监测

通过采用动态污点分析技术, 对 Android 内核源码进行定制, 编译和移植, 实现能实时监测报告第三方应用程序使用用户私有敏感信息情况的系统, 称之为 TaintDroid[10].

TaintDroid 自动给来自私有敏感信息源的数据标记标签, 且在敏感数据通过程序变量, 文件和进程间消息传播时, 传递地应用标签. 当被污染的数据通过网络传输, 或以其他方式离开系统, TaintDroid 则及时记录报告该数据的标签, 以及负责传输的应用程序和数据的目的地. 这种实时反馈为用户和安全服务提供了更深层次的洞察, 掌握移动应用程序的真正意图, 并有可能识别出行为不端的应用.

### 3.1 污点追踪方法

图 2 展现了智能手机污点追踪的方法架构图.
我们综合利用基于虚拟机的智能操作系统的结构特征(如, Android, Black- Berry, 和 J2ME-based phones), 同时采用具有清晰语义的细粒度的标签, 实现高效的, 全系统的污点追踪系统.

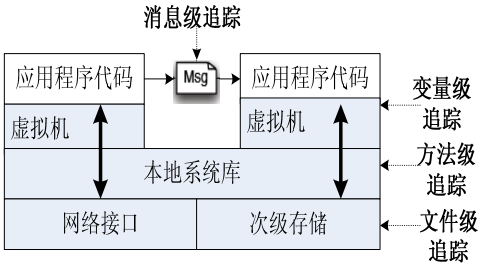


图 2 智能手机架构多层次的污点追踪方法

首先, 借助虚拟机解释器(VM Interpreter)在不可信的应用程序代码中提供变量级的追踪. 利用解释器提供的变量语义, 提供一个变量环境, 避免了在 x86 指令集中可能发生的污点爆炸. 通过跟踪变量实现了只需要维护数据的污点标记而不考虑代码的污点标记. 其次, 在应用程序间采用消息级追踪(message-level tracking). 在消息上追踪污点, 而不是在消息中的数据上追踪, 这样最大的减小了 IPC 的损耗, 从而扩展了全系统的分析. 再次, 针对系统提供的本地库, 使用方法级的追踪(method-level tracking). 最后, 使用文件级的追踪确保永久的信息保守地保留它们的污点标记.

### 3.2 TaintDroid 架构

TaintDroid 是在 Android 上实现多粒度污点跟踪的方法. TaintDroid 在 VM 解释器内使用变量级追踪. 多个污点标记(taint marking)存储为一个污点标签(taint tag). 当应用程序执行本地方法时, 变量的污点标签在返回时打上补丁. 最后, 污点标签被分配到 parcels, 通过 binder 进行传播.

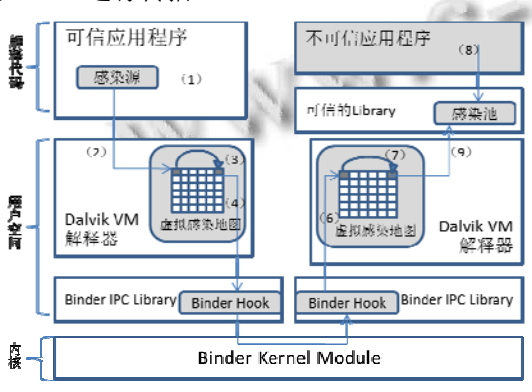


图 3 TaintDroid 的架构

图 3 描述了 TaintDroid 的架构. 在可信的应用程序内信息被感染(1), 且具备充足的情景环境(如, location provider). 污点接口调用连接 Dalvik VM 解释器的本地方法(2), 把特定的污点标记存储到虚拟污点地图(virtual taint map). 当可信应用程序使用污点信息时, Dalvik VM 根据信息流的规则传送污点标签(3). 每一个解释器实例都同时传送污点标签. 当可信应用程序在 IPC 事件中使用污点信息时, 改进的 binder 库(4)保证信息包(parcel)有一个可以反映所有包含数据的污点标记组合的污点标签. Parcel 明显地通过内核传输(5), 被远程不可信的应用程序接收. 这里需要注意的是只有解释过的代码才是可信的. 改进的 binder 库从 parcel 中恢复污点标签, 且分配给从 parcel 读取的所有值(6). 远程 Dalvik VM 实例同样地传送污点标签给不可信应用程序(7). 当不可信应用程序调用指定为污点池(taint sink)的库时(8), 如, 网络发送, 该库检索问题数据的污点标签且报告该事件.

### 3.3 TaintDroid 在模拟器上运行

TaintDroid 编译成功后, 在模拟器上启动, 运行被测试的应用程序后, 启动 logcat, 查看的监控记录. 该实验采用测试软件 TaintDroidTester.apk 进行试验, 如截图 4 所示, TaintDroid 监控到 TaintDroidTester.apk 应用程序, 分别发送带有 0x400, 0x1000,0x8 污点标签的敏感信息到 IP 地址: 58.83.143.22. 通过查看 TaintDroid 源码路径下的 dalvik/vm/interp/Taint.h 文件, 可以确定这些被发送的敏感信息分别为 IMEI,SIM 序列, 和手机号码.

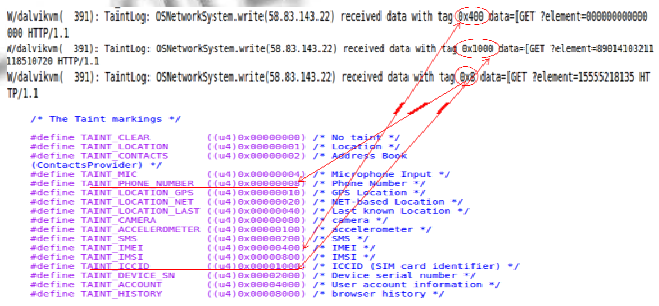


图 4 TaintDroid logcat 截图

## 4 Android平台上的隐私控制实现

在监测应用程序使用用户私有敏感信息的基础上, 增加阻断泄露(exfiltration blocking), 阻断被敏感信息污染的输出通信, 和数据遮蔽(data shadowing), 建立一个新的隐私控制系统 AppFence.

当应用程序请求访问敏感数据, 但用户不希望它

访问该数据, 于是 AppFence 用无害的阴影数据替换该数据. 例如, 一个应用程序请求通讯录, 但获取的是一个不包含联系人的阴影数据, 只包含一些用户认为不敏感的真实记录或虚构的阴影条目. AppFence 可使用户扣留被恶意应用程序请求的数据, 以防它用来执行不必要的广告功能, 确保数据只用于用户期望的用途; 同时也可以阻断应用程序从设备泄漏数据的通信.

### 4.1 技术实现

#### 4.1.1 数据遮蔽

为施加隐私控制于应用程序之上, AppFence 修改了核心库和 Android 框架层. 图 4-1 显示了 Android 架构中被修改以实现遮蔽的组件.

Android 应用程序利用文件系统访问相机, 麦克风, 和日志. 当应用程序尝试打开这些资源时, 我们提供一个打开空文件的假象. 同样的, 我们也可以通过返回一个数据空集来遮蔽浏览器数据(历史记录和书签), SMS/MMS 消息, 订阅信息, 通讯录, 账号信息, 和日历条目.

当应用程序请求设备位置时, 我们返回一个固定的坐标位置: 37.421265, -122.084026.

当应用程序请求设备的电话状态, 我们构建一个固定的手机状态: 手机号码(1 650 623 4000)和特定应用程序的设备 ID.

#### 4.1.2 阻断泄漏

为检测和阻断网络交通, 我们修改了 Android 网络协议层的 Java 代码和本地代码. 图 5 显示了为阻断泄露装备或创建的主要模块.

为阻断泄露数据, 拦截网络电话堆栈以实现 1)关联域名和公开的套接字, 2)检测被污染的数据什么时候被写入套接字. 当输出缓冲包含被污染的数据时, 我们终止缓冲,选择后面两个动作中的其中一个: 我们可以秘密地丢弃冒犯的消息, 误导应用程序指示缓冲区已发送; 或者, 公开地仿效 OS 应用程序可能遇到的行为, 例如由于设备进入飞行模式缓冲区被丢弃(禁用所有无线连接).

## 5 结束语

通过对动态污点分析技术理论的研究, 提出该技术的广义的理论框架; 其次, 融合动态污点追踪技术

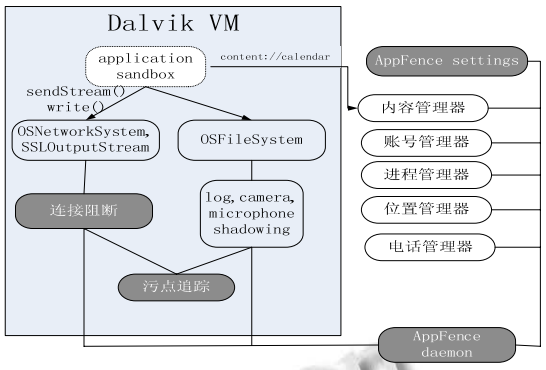


图 5 AppFence 系统架构图

和隐私控制技术，实现 Android 应用程序误用用户私有敏感信息的监测和控制. 当然该系统还存在许多的缺陷，这需要进一步研究，根据需求增加标签的数量和标签的粒度，支持基于控制流的传播，采用更复杂的影子遮蔽策略，突破现有的实现和污点源的局限性，更加精确的禁止第三方本地库等等, 促使系统功能更加完善, 稳定运行于不同版本的 Android 系统中.

## 参考文献

1 Hornyack P, Han SY, Jung JY, Schechter S, Wetherall D. These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications. ACM, 2011:1–6.

2 Leek T, Baker G, Brown R, Zhivich M, Lippmann R. Coverage Maximization using Dynamic Taint Tracing. [Technical Report] TR-1112. MIT Lincoln Laboratory. 2007: 2–3.

3 Masri W, Podgurski A, Leon D. Detecting and debugging insecure information flows. International Symposium on Software Reliability Engineering (ISSRE 2004). 2004. 198–209.

4 Kong J, Zou CC, Zhou H. Improving software security via runtime instruction-level taint checking. ASID'06, Proc. of the 1st Workshop on Architectural and System Support for Improving Software Dependability. 2008. 18–24.

5 Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. Proc. of the Network and Distributed System Security Symposium (NDSS 2005). 2005. 55-59.

6 Qin F, Wang C, Li Z, seop Kim H, Zhou Y, Wu Y. LIFT: A

low-overhead practical information flow tracking system for detecting security attacks. MICRO ’06: Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. 2009. 135–148.

7 Pietraszek T, Berghe CV. Defending against injection attacks through context-sensitive string evaluation. Proc. of Recent Advances in Intrusion Detection (RAID 2005). 2005. 3–8.

8 Nguyen-Tuong A, Guarnieri S, Greene D, Shirley J, Evans D. Automatically hardening web applications using precise tainting. 20th IFIP International Information Security Conference. 2005. 3–5.

9 周凌.基于信息流的动态污点分析技术研究[学位论文].成都:电子科技大学, 2010:24–25.

10 Enck W, Gilbert P, Chun BG, Cox LP, Jung J, Mcdaniel P. Sheth AN. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. Proc. of the USENIX Symposiumon Operating Systems Design and Implementation. 2010. 1–7.