第 38 卷 第 23 期
Vol.38 No.23

计 算 机 工 程
Computer Engineering

2012 年 12 月
December 2012

• 专栏 •

文章编号：1000—3428(2012)23—0001—06 文献标识码：A 中图分类号：TP309

一个面向 Android 的隐私泄露检测系统

杨广亮[1,2a,2b]，龚晓锐[2a,2b]，姚 刚[1]，韩心慧[2a,2b]

(1. 中国科学院研究生院信息安全国家重点实验室，北京 100049；
2. 北京大学 a. 互联网安全技术北京市重点实验室；b. 计算机科学技术研究所，北京 100871)

摘 要：针对 Android 软件中存在的用户隐私信息泄露问题，基于动态污点跟踪技术实现 TaintChaser 自动化检测系统。该系统能对软件中存在的用户隐私信息泄露行为进行细粒度的跟踪，实现对手机软件规模化自动化的检测与分析。利用该系统对 28 369 个 Android 程序进行检测，结果表明，有 24.69%的程序可能存在泄露用户隐私信息的行为。

关键词：动态污点跟踪；Android 系统；隐私泄露；自动化测试；恶意软件

A Privacy Leakage Detection System for Android

YANG Guang-liang[1,2a,2b], GONG Xiao-rui[2a,2b], YAO Gang[1], HAN Xin-hui[2a,2b]

(1. State Key Laboratory of Information Security, Chinese Academy of Sciences, Beijing 100049, China;
2a. Beijing Key Laboratory of Internet Security Technology; 2b. Institute of Computer Science and Technology, Peking University, Beijing 100871, China)

【Abstract】To detect user's privacy leakage in Android software, this paper proposes an automated detection system based on dynamic taint tracking, called TaintChaser. TaintChaser can detect behaviors of user's data leakage in Android applications under test at a fine granularity, and the system also can analyze and test massive Android software in the automatic way. It uses TaintChaser to automatically analyze 28 369 popular Android applications and finds that 24.69% of them may leak user's privacy.

【Key words】dynamic taint track; Android system; privacy leakage; automated test; malware

DOI: 10.3969/j.issn.1000-3428.2012.23.001

1 概述

最近几年，Android 系统的普及十分迅速。2011 年第 3 季度 Android 操作系统在全球智能手机市场中的占有率已经超过 50%，比 2010 年同期增加了 27.2%。但随着 Android 系统的逐渐流行，手机中用户隐私数据(通讯录、个人信息数据、电子邮件等)的泄露问题也更加突出。文献[1]对电子市场中的 50 个软件进行了测试，发现 34%的软件存在泄露隐私的行为。但是测试的样本数量都很少，同时 Android 电子市场新的应用软件快速增加，这些研究工作并不能完全反映 Android 电子市场中应用软件泄露隐私信息的真实情况。因此，本文提出并实现一种自动化的隐私泄露检测工具，能够对国内主流的电子市场进行规模化的自动检测。

2 相关工作

2010 年 Enck 等人[2]在 Android2.1 操作系统上实现了动态污点跟踪系统 TaintDroid。2011 年一系列基于 TaintDroid 的系统也被提出，包括 AppFence[1]和 Mock Droid[3]等。它们都对 TaintDroid 进行了不同程度的扩展，但是 TaintDroid 固有的一些缺陷并没有得到解决。这些缺陷包括以下 5 个部分：

(1)能够检测的隐私数据类型不够全面，包括了 IMEI、电话号码、地理位置信息、相册、录音，而通信录、短信息等重要信息并没有得到检测。

(2)检测的隐私泄露点不够多，仅仅包含一般网络通信(socket)；而蓝牙、短消息等通信方式没有被监测。

(3)字符串跟踪粒度不够细，可能导致污点跟踪过程中可能会产生污点的过度扩散，从而产生误报。

在 TaintDroid 中，字符串对象是当作一个整体来进行污点标记的。当将一个带有污点的字符串拆分开时，它所有的子字符串都会带有污点。这样会导致污点的过度扩散。例如，假设字符串 A“This is a test”是不带污点的正常数据，而字符串 B 是被标记了污点的电话号码“12345

基金项目：国家自然科学基金资助项目(61003217)；国家发展和改革委员会 2010 年信息安全专项基金资助项目(发改高技[2010]3044 号)

作者简介：杨广亮(1987－)，男，硕士研究生，主研方向：网络安全，恶意代码分析；龚晓锐，学士；姚 刚，博士；韩心慧(通信作者)，高级工程师

收稿日期：2012-02-07 **修回日期：**2012-03-11 **E-mail：**hanxinhui@pku.edu.cn

678901”，将字符串 A 和 B 拼接起来构成字符串 C 后，新的字符串 C 则是带有污点的，其中它的内容是“This is a test 12345678901”。然后取出字符串 C 的前 4 位，得到新的字符串 D“This”，此时 D 也是带有污点的。如果将字符串 D 通过隐私泄露点发送出去时，由于 D 是污点数据，因此这种行为是“隐私泄露”。但是从整个数据的流程来看，字符串 D 不应该是带有污点的，这实际上是个误报。

(4)不能提供程序测试过程中所执行的路径信息。TaintDroid 系统并不能提供任何被测试软件在其测试过程中所执行的路径信息，无法区分多次测试是否执行了相同的程序路径，导致测试带有非常大的盲目性。

(5)测试时需要人工的参与，不利于自动化测试。

3 现阶段用于检测隐私泄露的技术及其缺陷

现有用于检测用户隐私数据泄露的技术按照是否需要执行程序可分为 2 类：静态方法和动态方法。静态方法主要有控制流分析、数据流分析及结构分析等[4]。但是因为 Android 程序基本上都是 Java 代码，所以 Android 程序会存在大量隐式函数调用(例如虚函数等)。而对于这类调用，静态分析并不能进行有效的处理。同时，通过静态分析虽然可以得到程序中隐私泄露的具体执行路径，但是并不能确认这条路径是否真正可以被执行，只能通过动态的方法加以验证。

动态方法则有传统的沙箱技术以及动态污点跟踪技术等。沙箱技术是一种程序的隔离运行机制，目前广泛地应用于软件测试、病毒检测等领域[5]。利用沙箱技术进行隐私泄露检测时，需要监测系统中读取用户敏感信息、网络通信等重要的接口。当程序调用这些接口时，能及时地给予记录。通过这种方法，可以很好地检测到程序是否读取了用户敏感的信息和是否通过网络向外部发送数据。但由于系统的监测点不是连续的，它们之间缺乏必然的逻辑关系和上下文关联性，因此使用沙箱技术并不能准确地判定程序是否泄露了用户的隐私数据。

而与沙箱不同，动态污点跟踪技术则是连续跟踪数据的流向，有效地解决了沙箱技术上下文信息缺乏关联的缺陷。目前动态污点跟踪技术广泛用于漏洞挖掘等方面[6-8]。将动态污点跟踪应用于检测隐私数据泄露时，将隐私数据作为污点数据，然后跟踪其传播，当检测到这些污点数据通过网络等方式被发送出去时，则可以判定该程序存在隐私泄露问题。

对于检测程序中存在的隐私数据泄露行为，动态污点跟踪是一种非常有效的方法。基于该方法，本文实现了一个 TaintChaser 系统，对隐私信息进行细粒度的跟踪，能产生被测试程序在测试过程中所执行的路径信息，并基于 TaintChaser 实现了自动化检测系统。

4 动态污点跟踪技术的基本原理

动态污点跟踪技术的基本原理如图 1 所示。其中，程序 A 和程序 B 下面方框中的 6 条线分别代表该程序内运行的进程，曲线代表该线程中当前没有包含污点数据；虚线则代表该线程存在污点数据，不同虚线代表包含不同类型的污点数据。

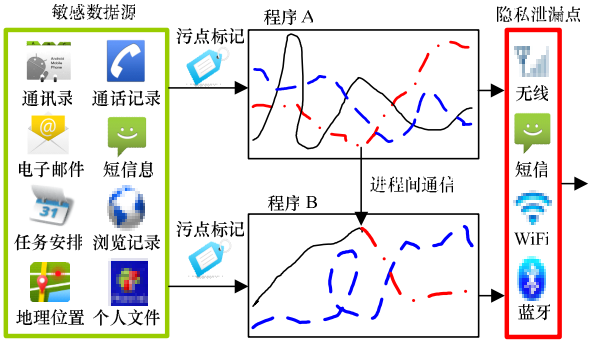

图 1 动态污点跟踪技术原理

当检测到程序在读取用户隐私信息时，将读取到的隐私数据标记污点。而当程序对带有污点的用户敏感数据进行操作时，能够对该操作进行相应处理，保证污点能够跟随隐私数据传播下去。当 2 个程序之间进行通信时，污点也能跟随数据正常跟踪下去。例如，图中程序 A 中的一个进程(点虚线)将带有污点的数据发送给程序 B 中的一个线程后，污点在该线程中能继续跟踪着数据传播下去。当程序把带有污点的数据通过隐私泄露点发送到外面时，则实时地记录该行为。最后分析得到的日志，来判断程序是否存在隐私泄露。如果该日志中存在程序将污点数据发送出去的记录，则判定程序存在隐私泄露的问题。

5 Android 系统

5.1 Android 系统现有的保护机制及其缺陷

Android 是基于 Linux 系统、针对手机平台开发的开源操作系统。Android 操作系统本身提供了一系列用于隐私数据保护的机制。Android 扩展了 Linux 系统使得每一个应用程序都可以以不同的身份标识(即 Linux 原有的 uid 和 gid)运行，从而保证程序运行环境的相对独立性。Android 提供了一种权限系统，该系统提供了对用户资源(手机设备信息、网络资源等)进行访问控制的机制。在该权限系统中，手机中的重要资源被划分成几类，每一类资源都对应一种权限；而当程序对某类资源进行访问时，必须当拥有对应的权限才能获得执行授权。

但是这种保护机制存在很大的局限性，不能有效防止程序泄露用户的隐私信息。这种用于对资源的访问控制的权限系统实现的粒度过于粗糙，并不能达到预期的效果[9]。该权限系统具体实现不够灵活。一个程序所拥有的权限完全是在它被安装时确定的；而当安装某一个程序时，对于一个程序权限的选择，Android 也仅仅提供了 2 种：全部满足程序所要求的权限或者全部否定，而不是让用户根据自己的实际情况只选择满足部分权限。程序如果在安装阶段拿到它所需要的权限，那么在运行阶段，就能任意泄露用户隐私而不会受到系统限制。综上，Android 本身自带的机制并不能有效地防止程序泄露用户隐私数据。

5.2 体系结构

Android 操作系统的体系结构[10]如图 2 所示。

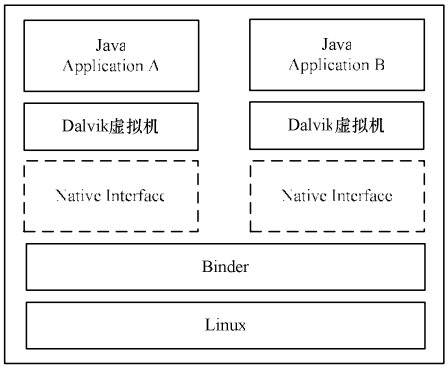

图 2 Android 操作系统的体系结构

最上层是实现了各种功能的 Java 程序；下一层则是 Dalvik 虚拟机，它是 Google 专门为 Android 系统设计开发的 Java 解释器，Java 程序在它的解释下得以运行；再下层是 Native Interface，它是可由上层 Java 程序调用本地代码库；Binder 层则是 Android 系统提供的一种轻量级的进程间通信机制；最底层则是经过修改过后的 Linux 系统，提供了与手机硬件交互的相关接口。

5.3 Dalvik 虚拟机

本质上，Dalvik 虚拟机是一种 Java 虚拟机，但是它又与一般的 Java 虚拟机有很大的不同。不同之处主要有：(1)Dalvik 虚拟机与 Java 虚拟机体系结构是不同的。Dalvik 虚拟机是基于虚拟寄存器而实现的，而 Java 虚拟机则是基于堆栈的。(2)Dalvik 虚拟机的指令集(dalvik 字节码)与 Java 虚拟机(Java 字节码)也是完全不同的。Dalvik 虚拟机是整个系统中较为核心的部分。在动态污点跟踪过程中，大部分污点的传播是在这一部分进行的。

5.4 JNI

JNI(Java Native Interface)是一种用来让 Java 程序与本地代码库交互的机制。通过这种机制，Java 代码可以方便地调用其他语言(一般是 C 和 C++)写的库。而对于动态污点跟踪来讲，当进行 JNI 调用时，代码会从 Java 层转到了 C/C++层，这会影响污点数据的正常跟踪，需要对涉及到的 C/C++库进行相应的处理。

6 TaintChaser 系统的实现

与 TaintDroid 相比，本文提出的 TaintChaser 系统对更多的用户隐私信息进行了监控，不仅检测了 IMEI、电话号码、地理位置信息、相册、录音，还将通信录、短信息、通话记录、电子邮件等重要信息作为污点源进行了检测。同时该系统通过检测内存中每一个字节的内容，实现了一种粒度更细的污点跟踪方式；检测了更多的隐私泄露点(socket 通信、https 加密通信、短信和蓝牙通信)；能够给出被检测程序执行详尽的路径信息；并能对程序进行自动化的测试。

6.1 污点数据

TaintChaser 系统可以检测绝大部分用户隐私信息：手机设备标识号，电话号码，地理位置信息，电子邮件，通信录，短信息，日程安排以及浏览器历史记录。

因为获取这些隐私信息的方式是不同的，所以进行污点标记时需要分别进行处理。其中手机设备标识号、电话号码、地理位置信息都是由 Android 系统中相关的服务进程提供的，需要对这些服务进程进行相关处理。其污点标记流程如图 3 所示。在数据读取时，程序作为客户端通过 binder 向相应的服务进程发出数据请求；服务端取得隐私数据；TaintChaser 系统将这些数据标记污点，从而保证程序拿到的数据都是标记了污点的。

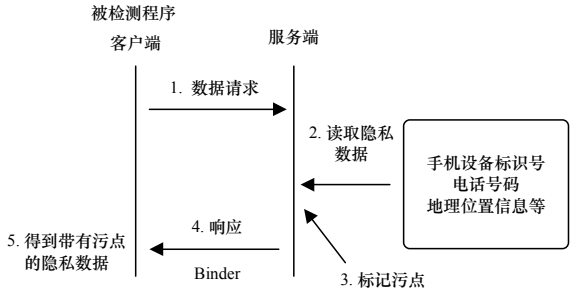

图 3 TaintChaser 系统中的污点标记流程 1

电子邮件、通信录、短信息、日程安排、以及浏览器历史记录存储于数据库中。对于这些数据的污点标记，需要对读取这些数据库的行为进行相关处理。其中从数据库中读取数据的一般流程是先需要得到数据库句柄 cursor，然后通过利用该 cursor 调用相关函数(getString 等)读取数据库中的内容。

对存于数据库的用户敏感信息标记污点过程如图 4 所示，当程序获取这些存储用户隐私信息数据库的句柄 cursor 时，将该 cursor 标记污点。

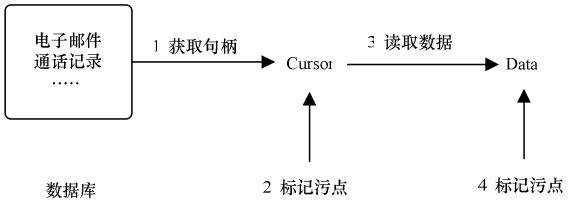

图 4 TaintChaser 系统中的污点标记流程 2

在程序后面的运行中，如果检测到程序通过这个带有污点 cursor 读取具体数据内容时，那么读取到的所有数据都会被标记污点。通过这种方式，最终实现对电子邮件、短消息等这类存储于数据库隐私信息的污点标记。

6.2 污点存储

与 TaintDroid 中污点的存储有很大不同，TaintChaser 使用一种细粒度方式来描述污点：内存中的每个字节都对应着一个污点。通过这种方式，在对污点数据进行跟踪时，可以进行更为准确地跟踪，大大减少了污点过度扩散的机率。其污点的存储结构如图 5 所示，由于 Android 系统是 32 位的，污点存储表以二级地址索引方式组织。这样存储方式可以在最大化节省内存空间的情况下，实现快速查找内存对应的污点。

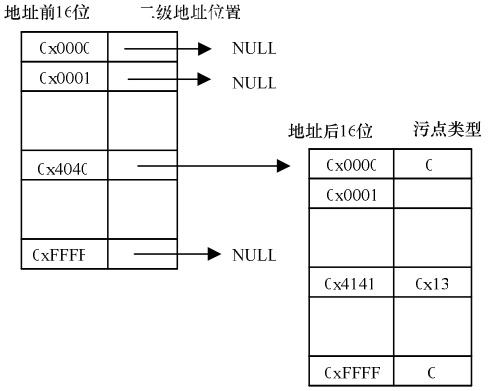

图 5 TaintChaser 系统中污点的存储结构

6.3 污点跟踪

Android 程序都是通过 Dalvik 虚拟机的解析而得以执行的。TaintChaser 系统通过处理 Dalvik 虚拟机中所有相关指令，来实现对污点数据的跟踪。而这需要从以下 3 个方面进行处理：

(1)一般指令：主要包括一系列对变量的基本操作(赋值和加减乘除等)和函数调用。对于变量的基本操作，如果涉及的操作数中包含污点数据，那么最后的结果也要标记污点。而对于函数调用，污点数据可能通过函数的输入参数和返回值进行传播。需要对 invoke 和 return 指令进行相关的处理。

(2)文件读写：主要指对文件的读写行为进行检测。当发现污点数据被写入到文件时则，对该文件标记污点；而当检测到带有污点文件内容被读取出来时，也能够实时地进行处理。

(3)进程间通信：当检测到进程间通信时，如果通信的内容中包含污点数据，能实时地跟踪污点数据地传播，保证污点可以在新的进程中继续跟随数据传播。

而对于 Android 应用程序中的 JNI 机制，因为程序会脱离 Dalvik 虚拟机进入了本地 C/C++库，此时系统不能进行正常的污点跟踪。为实现污点的正常的传播，通过 hook 相关函数方式来进行处理。

6.4 隐私泄露点

TaintChaser 系统检测隐私泄露点主要有 Socket 通信接口、https 加密通信接口、蓝牙和短消息等。对于每一类隐私泄露点来说，因为将数据发送出去的接口不是唯一的，所以需要对涉及到的所有函数都进行相关处理：首先根据发送数据的长度和它的内存地址，到污点存储表中检查该数据是否是污点数据，如果是，则将发送的数据内容和发送的目的地址等相关信息记录下来。

6.5 被测程序执行路径信息的输出

输出被测程序所执行过的路径信息，是通过处理 Dalvik 虚拟机中函数调用指令(invoke)来实现的。当执行 invoke 指令时，先将被调用的类和方法名打印出来即可。打印出来的信息里面就包含了被测程序调用关系。

但是因为被测试程序在执行过程中会调用大量系统的函数(例如界面渲染、进程间通信、各种事件处理等)，所以打印出来的结果不仅仅包括被测程序的执行路径，还会包含了大量系统函数调用信息。因此，需要加入过滤机制，将系统函数的调用信息等无用信息过滤掉。由于系统函数的类名和软件的类名是不同的，因此可以根据调用函数的类的名称来实现过滤。

首先在程序测试之前，需要提取被测试程序的相关类名，并将其导入到指定的配置文件中；然后在其测试时，当执行 invoke 指令时，将被调用函数的类名与配置文件中的内容进行匹配，如果相匹配，则将被调用函数相关的信息输出，否则直接抛弃；这样最终可以得到被测试程序所执行过的路径信息。

6.6 自动化测试程序

自动化测试程序需要实现自动安装程序到系统，然后能自动将该程序在系统中启动，并能在程序执行过程中自动向程序发送一系列的事件。其中，自动安装和启动可利用 Android 系统提供的工具 adb 和 am 来实现。

在自动化地测试程序之前，需要进行一些预处理。首先，解压缩被测试程序，从文件 AndroidManifest.xml 中读取程序相关信息。然后，根据程序的文件目录结构推测出被测试程序类名，将类名组织成文件导入到指定的目录中，以便用于输出被测程序所执行过的路径信息。这些准备工作完成后，就可以利用 adb 工具将被测试程序安装到系统中，并将其启动，开始测试。而在测试过程中，使用 am 命令向程序发送一系列的事件。

7 基于 TaintChaser 的自动化测试系统

7.1 开发环境

基于 TaintChaser 自动化测试系统是在北京大学移动互联网安全威胁监测平台上实现的，该平台提供了完备的用于自动化测试的相关接口，并有大量真实的 Android 应用软件样本可供分析。

TaintChaser 系统在由 Android 系统提供的模拟器运上行。该模拟器比真实手机在性能上有所不足。这主要是因为在程序执行的过程中，模拟器需要花费额外的时间把 ARM 的指令转化为 x86 指令；同时在渲染画面时，模拟器由于在硬件方面也缺少相应地对画面进行加速处理机制，所以比真实手机慢很多。但是由于模拟器可以运作在一般 pc 上，因此可以通过大规模并行部署的方式来弥补其性能上的缺陷。TanitChaser 是基于 KVM 虚拟机来进行大规模部署，通过统一的系统调度来实现并行检测。

7.2 自动化测试系统的实现

基于 TaintChaser 的自动化测试系统结构如图 6 所示。它主要由安装 kvm 虚拟系统的宿主机(真实计算机)，以及一系列虚拟机(kvm#1、kvm#2 等)组成，其中，kvm#1，kvm#2 等虚拟机用于自动检测程序的隐私泄露问题，是在宿主机上克隆出的一系列完整测试环境。整个系统主要由 3 个模块组成：Schedule，TaintChaser 和 Restore。其中，

Schedule 模块运行在虚拟机中，主要用于将所在虚拟机注册到北京大学移动互联网安全威胁监测平台上，并从该平台上获取测试任务和被测试的程序，并将得到的程序交给 TaintChaser 进行分析处理，然后把处理后的结果传回给监测平台；TaintChaser 运行在各个虚拟机中，完成对程序进行自动化的检测；而 Restore 模块则运行在宿主机上，主要负责将完成测试了的 kvm 虚拟机恢复到它原始干净的工作环境。

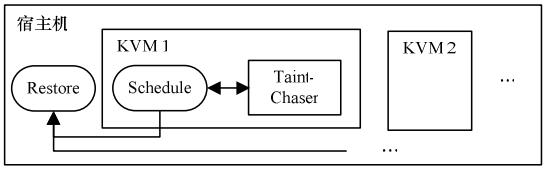

图 6　基于 TaintChaser 的自动化测试系统结构

以测试系统虚拟机 kvm#1 为例，整个工作流程如下：

(1)当测试系统启动后，schedule 模块会自动注册到移动互联网安全威胁监测平台，然后等待监测平台给分配检测任务。

(2)当 schedule 接受到任务时，根据任务的内容(程序 id，程序存放位置)，从指定位置下载待测试的程序到本地，并将该程序交给 TaintChaser 进行检测分析。

(3)TaintChaser 在检测完程序后，会将检测时打印日志传给 schedule 模块。

(4)schedule 模块则会分析所得到的日志，并把把分析的结果写回到监测平台中。

(5)schedule 调用宿主机上的 restore 模块恢复虚拟机 kvm#1 到原始的工作环境。然后重新进行步骤(1)，开始进行新一轮的测试。

8　测试结果及评估

实验的测试样本都是由北京大学移动互联网安全威胁监测平台提供的，主要来源于国内的各个 Android 电子市场。具体情况见图 7，样本主要从电子市场 appchina.com、eoe marke.com 和 goapk.com 等获得，其中总样本数量为 28 369。

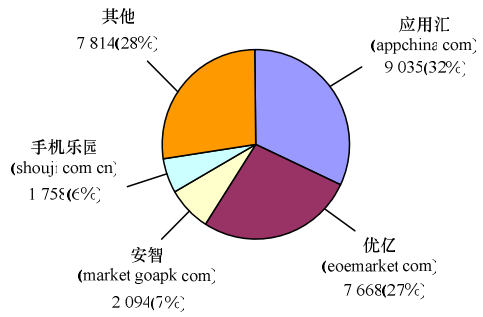

图 7　Android 测试程序样本来源

使用基于 TaintChaser 的自动化测试系统对这些 Android 样本程序进行了大规模自动化的分析和检测，最终发现有 7 004 个(占总数 24.69%)程序可能存在泄露用户隐私信息的行为，如表 1 所示。

表 1　泄露用户隐私信息的程序数统计

泄露的隐私数据类型	程序数目	占总量百分比/(%)
设备标识号 IMEI	6 373	22.46
电话号码	1 952	6.88
地理位置信息	482	1.70
设备序列号 serial number	229	0.81
短信息	3	—

随后从泄露隐私信息每一类软件中分别随机地取出了 30 个软件进行人工分析，对隐私信息泄露的原因进行了更加深入的分析。

(1)IMEI

在 30 个测试软件中，有 22 个软件会将 IMEI 发送给广告服务商，而剩余的 8 个软件则会发送 IMEI 到各自的软件开发商。其中软件向广告服务商发送用户的隐私信息，大都是由于软件开发者处于盈利的目的，在软件中植入了广告而导致的。而在软件中插入广告都是通过在软件代码中插入第三方相应广告插件来实现的。

(2)电话号码

对于电话号码，有 19 个测试软件会将电话号码发送给广告服务商，而剩余的 8 个软件则会发送到各自软件开发商。

(3)地理位置信息

30 个测试软件中，有 5 个软件属于正常应用，16 个软件会将地理位置信息发送给广告服务商，剩余 9 个软件会将地理位置信息发送给相应软件开发商。其中，5 个正常使用地理位置信息的软件中，有 2 个软件提供正常的地图服务，剩余 3 个软件则是利用 gps 位置坐标来查找附近的美食、电影等。

(4)设备序列号

与上述几个隐私信息有很大的不同，30 个测试软件中，并没有软件会将设备序列号发送给广告服务商，而是全部发送给了各自的软件开发商。

软件泄露隐私信息情况如表 2 所示。

表 2　软件泄露隐私信息情况

泄露的隐私信息类型	目的地	目的地域名	软件数量
IMEI	广告服务商	youmi.net	16
		wiyun.com	5
		mobclix.com	1
	软件开发者	—	
电话号码	广告服务商	oumi.com	16
		sosceo.com	3
	软件开发者	—	8
地理位置信息	广告服务商	youmi.net	8
		lsense.cn	6
		mobclix.com	2
	软件开发者	—	9
设备序列号	软件开发者	—	30

然后对国内各个 Android 电子市场中软件的隐私泄露情况进行了统计，具体结果如表 3 所示，发现国内各个电子市场都不同程度存在泄露用户隐私信息的软件。

表 3 各个电子市场泄露隐私的软件数量

电子市场名称	软件数量	泄露隐私的软件数量	百分比/(%)
goapk.com	2 094	795	37.97
eoemarket.com	7 668	2 794	36.44
appchina.com	9 035	1 690	18.71
Shouji.com.cn	1 758	187	10.64

同时通过该自动化检测系统，也发现了一些会泄露用户隐私信息的第三方插件。以恶意插件“com.nl”为例，该插件会在手机启动时开始运行，首先向某僵尸网络发送请求命令，然后根据得到的命令进行一些相应处理，例如自动从指定服务器下载安装一些比较流行的软件(例如 qq、人人网客户端等)，发送扣费短信等。并且该插件会拦截地址是“10658166”或内容中包含有“83589523”、“客服电话”、“元/条”、“元/次”、“本次 1 元”和“本次 2 元”等字符串的短消息。其中，在与僵尸网络通信过程中，该插件会泄露用户的电话号码和 IMEI 信息。

随机地从样本库中挑选 50 个软件(占总量的 0.18%)，用 TaintDroid 和 TaintChaser 分别进行检测，发现 Taint Droid 检测出来 29 个软件可能存在隐私信息的泄露，而 TaintChaser 检测到了 32 个。

对于国内电子市场上软件隐私泄露的问题，究其原因可以归纳为以下几点：

(1)国内 Android 软件开发人员缺乏保护用户隐私数据的意识，开发的软件往往会泄露用户的隐私数据。

(2)也有一部分国外的 Android 软件在被“汉化”的时候，被植入了一些第三方插件(例如广告插件等)，这样可能会导致用户隐私数据的泄露。例如，对于同一版本的游戏“愤怒的小鸟”，英文版不存在泄露用户隐私信息的问题，而国内电子市场提供的相关“汉化”版，则被插入了会泄露用户地理位置的 admob 广告插件。

(3)也有的电子市场出于收集用户信息、分析用户行为的目的，在自己市场中比较流行的软件中植入一些可能会泄露用户隐私信息第三方插件。

(4)国内 Android 电子市场对软件上传管理不够规范，没有进行相关的软件审查的制度和应对机制。

9 结束语

随着 Android 手机平台的快速发展，Android 软件中隐私泄露的问题也更加严重。本文基于动态污点跟踪技术，提出并实现了自动化、规模化的检测系统 TaintChaser，并利用该系统对国内电子市场中软件存在的隐私泄露问题进行了检测和分析。下一步将优化系统，提高系统的工作效率[11]，并更加深入地分析泄露用户隐私数据的行为模式。

参考文献

[1] 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]//Proc. of CCS’11. Chicago, USA: [s. n.], 2011.

[2] Enck W, Gilbert P, Chun B, et al. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones[C]//Proc. of OSDI’10. Vancouver, Canada: [s. n.], 2010.

[3] Beresford A R, Rice A, Skehin N, et al. MockDroid: Trading Privacy for Application Functionality on Smartphones[C]//Proc. of the 12th Workshop on Mobile Computing Systems and Applications. Phoenix, USA: [s. n.], 2011.

[4] Enck W, Octeau D, McDaniel P, et al. A Study of Android Application Security[C]//Proc. of the 20th USENIX Security Symposium. San Francisco, USA: [s. n.], 2011.

[5] Goldberg I, Wagner D, Thomas R, et al. A Secure Environment for Untrusted Helper Applications(Confining the Wily Hacker)[C]//Proc. of the 6th USENIX UNIX Security Symposium. San Jose, California, USA: [s. n.], 1996.

[6] Sabelfeld A, Myers A C. Language-based Information-flow Security[J]. IEEE Journal on Selected Areas in Communications, 2003, 21(1): 5-19.

[7] Ligatti J, Bauer L, Walker D. Edit Automata: Enforcement Mechanisms for Run-time Security Policies[J]. International Journal of Information Security, 2005, 4(1-2): 2-16.

[8] Wang Tielei, Wei Tao, Gu Guofei, et al. TaintScope: A Checksum-aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection[C]//Proc. of the 31st IEEE Symposium on Security & Privacy. Berkeley, USA: [s. n.], 2010.

[9] Enck W. Defending Users Against Smartphone Apps: Techniques and Future Directions[C]//Proc. of ICISS’11. Kolkata, India: [s. n.], 2011.

[10] 曾健平，邵艳洁. Android 系统架构及应用程序开发研究[J]. 微计算机信息, 2011, 27(9): 1-3.

[11] Gilbert P, Chun B, Cox L P, et al. Vision: Automated Security Validation of Mobile Apps at App Markets[C]//Proc. of MCS’11. Washington D. C., USA: [s. n.], 2011.

编辑 顾姣健