Android 应用权限检测技术研究

雷 磊　　胡 勇
(四川大学电子信息学院　成都　610064)
(jellyleiscu@163.com)

Research on Android Application Permission Monitor

Lei Lei and Hu Yong
(*College of Electronic and Information Engineering, Sichuan University, Chengdu* 610064)

Abstract　With the rapid rise of mobile Internet in recent years, smart phones, especially based on Android system, also developed rapidly. The issues of Android system become increasingly prominent. Though Android system provides a relatively complete security mechanism, its "All-Or-None" application authorization mode, as well as the permission management mode in which users cannot change their permissions after the application is installed, exists security risks. Therefore, this paper proposed a method based on the static permission analysis and code injection method, to achieve the target for real-time monitoring of sensitive permission. Experiments show that this method can effectively monitor the use of permissions.

Key words　Android; malicious application; static analysis; repackage; permission monitor

摘　要　随着近年来移动互联网的快速兴起，智能手机，特别是基于 Android 系统的智能手机极速崛起. Android 系统问题日益突出，Android 系统虽然提供较完整的安全机制，但其"All-Or-None"的应用授权模式以及应用一旦安装，用户就无法更改其权限，这一权限管理模式存在安全隐患. 为此，提出一种基于静态权限分析，并通过重打包注入代码的方法，实现对目标应用敏感权限的实时监控. 实验证明，该方法能对权限的使用进行有效监控.

关键词　安卓；恶意应用；静态分析；重打包；权限监控

中图法分类号　TP309

2016 年 4 月，360 互联网安全中心发布《2015 年中国互联网安全报告》，报告指出 2015 全年，360 累计截获 Android 平台新增恶意应用样本 1 874.0 万个. 分别是 2013 年、2014 年的 27.9 倍、5.7 倍；日均截获数高达 51 342 个. 2015 全年，累计监测到 Android 用户感染恶意程序 3.7 亿次，分别是 2013 年、2014 年的 3.8 倍和 1.1 倍；日均感染量达 100.6 万次. 在所有手机恶意程序中，资费消耗类恶意程序的感染量仍然保持最多，占比高达 73.6%；其次为恶意扣费(21.5%)和隐私泄露(4.1%). Android 用户所面临的威胁日趋严重.

Android 系统虽然提供了较为完整的安全管理机制[1-3]，但是在权限管理方面还是有所不足. Android 权限采用一次性授予机制，应用所有权限是在安装过程中申请的，用户只能全部接受或者拒绝安装；并且用户无法对已安装应用的权限进行管理，这给用户带来了严重的安全隐患. 另一方面，该机制不能有效地防止权限升级攻击[4].

收稿日期：2017-01-12

在 Android 应用权限检测方面，已有较多研究. 文献[5-6]基于 Android 的广播机制，对电话、网络、短信等数据进行动态监控，一定程度上增强了系统的安全性. 文献[7]通过 Permission Manager 实现权限的预先配置管理，对所有待安装应用进行统一的权限限制，从而实现对应用权限一定程度的安全管理. 文献[8]依靠 Xposed 框架，并采用动态插桩技术跟踪敏感函数，实现对应用运行时函数调用过程的跟踪记录. 文献[9]提供一种优秀的类层框架，并采用 Field-Sensitive 组件分析出的权限 API 映射规则，提升了 Android 静态分析能力. Chen 等人[10]质疑 Android 的安全模型，指出当前 Android 权限模型不能满足用户的安全要求，并提出了一种与 Apex 相兼容的策略模型. 该方法主要针对 Apex 的安装程序包 Poly，应用在安装之前，需要事先分析检测其中的权限，并将分析结果通知用户，以便用户决定是否安装该应用. 该模型对本文提出的策略具有一定的指导作用.

以上文献，针对 Android 权限问题，提出了不同的解决方法，有一定的指导意义和研究价值，然而还是有自身的不足之处. 针对 Android 一次性权限授予机制的缺陷，本文提出了一种基于静态权限分析[11-12]，并通过重打包注入监测代码的方法监测应用权限的使用，优化了 Android 系统本身的权限管理机制. 为了方便用户对权限的更改，监测代码中还包含日志记录功能，并通过手机客户端向用户呈现. 图 1 为系统的流程图. 首先通过静态权限分析，分析出应用申请的所有权限；接着将分析的权限与权限函数映射表相匹配，得到映射后的函数；然后对相应函数注入监测代码并重打包应用[13]；最后将重打包的应用安装到手机端，判断是否能够对敏感权限进行有效地监控.

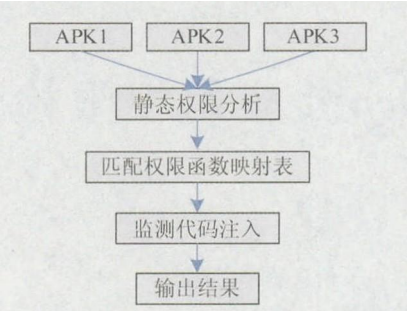

图 1 系统流程

1 关键技术

1.1 静态权限分析

Android 系统充分利用了 Linux 的用户权限管理方法，然而也有其创新之处. 开发者在开发应用时会在 AndroidManifest. xml 文件采用〈uses-permission〉前缀标明所有权限. Android 的每种权限都有一个独立的表示方式，如：

〈uses-permission android：name＝"android.permission. DELETE_PACKAGES" /〉；

〈uses-permission android：name＝"android.permission. INTERNET" /〉；

〈uses-permission android：name＝"android.permission. READ_PHONE_STATE" /〉；

〈uses-permission android：name＝"android.permission. INSTALL_PACKAGES" /〉.

本文采用批处理方式，调用 Apktool 工具，将目标应用反编译为包含 AndroidManifest. xml 在内的多个文件，然后提取〈uses-permission〉后面的字符串，即获得目标应用申请的所有权限；本方案通过对大量样本进行静态权限提取并统计分析，选取出现次数最频繁的前 15 种权限，并整理出其在良性样本和恶意样本中出现的次数，如图 2 所示：

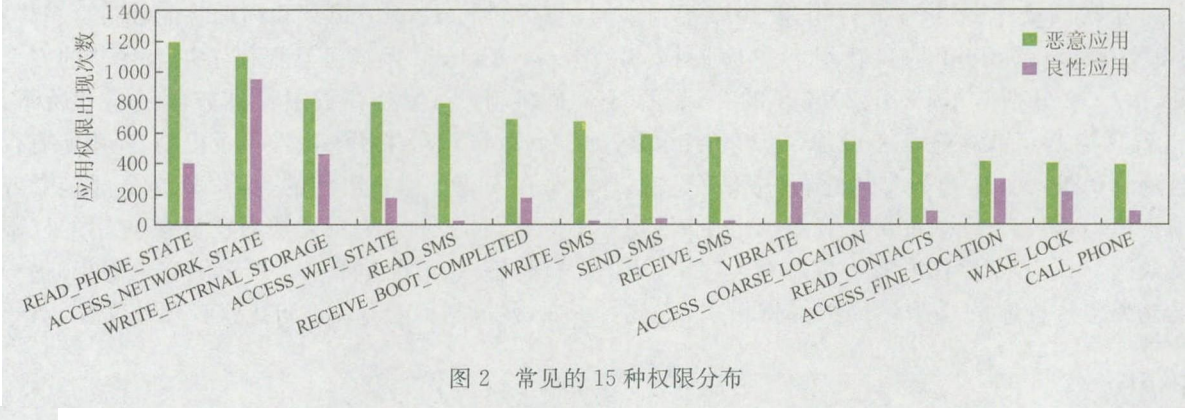

图 2 常见的 15 种权限分布

由图 2 可知，有些权限在恶意应用和良性应用中出现的频率基本相同，如获取网络链接的 ACCESS_NETWORK_STATE 权限；而另外一些权限在恶意应用中出现的频率远高于良性应用，如 READ_PHONE_STA-TE，WRITE_SMS 等，这类权限被本文标记为敏感权限. 为了体现区分度，本文计算出每一种权限在恶意应用和良性应用出现频率的比值，然后选取比值最大的前 10 个权限，作为恶意应用分析监测对象. 这 10 个权限如表 1 所示：

表 1　恶意应用监测权限

权限名	功能描述
android. permission. READ_SMS	读取短信
android. permission. READ_CONTACTS	读取联系人
android. permission. RECORD_AUDIO	允许录制音频
android. permission. CAMERA	允许拍照
android. permission. ACCESS_FINE_LOCATION	获取地理位置
android. permission. SEND_SMS	发送短信
android. permission. CALL_PHONE	打电话
android. permission. RECEIVE_SMS	接收短信
android. permission. READ_PHONE_STATE	读取手机状态
android. permission. INTERNET	网络连接

1.2　权限的函数映射

Android 系统预先封装一些函数，便于 Android 应用开发者调用这些函数实现某项功能. 同时应用要实现某项权限，也必定会调用相关联的函数. 比如 ACCESS_FINE_LOCATION（手机定位）权限：手机定位一般包含卫星（GPS）定位、基站定位、WiFi 定位，这几种定位都有具体的调用方法，Landroid/location/Location Manager;－>getLastKnownLocation 便是采用 GPS 定位时必须调用的函数，于是可以将 ACCESS_FINE_LOCATION 这项权限映射为函数 Landroid/location/Location Manager;－>getLastKnown-Location. 本文基于文献[8]提出的 InsightDroid 系统，同时结合恶意应用权限分布，绘制出恶意应用权限-函数映射表，如表 2 所示：

表 2　恶意应用权限-函数映射表

权限名	函数映射
android. permission. READ_SMS android. permission. READ_CONTACTS	Landroid/content/ContentResolver;－>query
android. permission. RECORD_AUDIO	Landroid/media/MediaRecorder;－>start Landroid/media/MediaRecorder;－>stop
android. permission. CAMERA	Landroid/hardware/Camera;－>open
android. permission. ACCESS_FINE_LOCATION	Landroid/location/Location Manager;－>getLastKnownLocation
android. permission. SEND_SMS	Landroid/telephony/SmsManager;－>sendDataMessage Landroid/telephony/SmsManager;－>sendTextMessage Landroid/telephony/SmsManager;－>sendMultipartTextMessage
android. permission. CALL_PHONE	Landroid/content/Context;－>startActivity
android. permission. RECEIVE_SMS android. permission. READ_PHONE_STATE	Landroid/content/Context;－>registerReceiver Landroid/content/Context;－>unregisterReceiver
android. permission. INTERNET	Ljava/net/URL;－>openConnection

1.3 监测代码注入

监控代码注入主要思想：对表 2 中提到的函数进行 2 次重打包，当应用试图调用敏感函数时，会被监测代码先捕获，接着监测代码立即检查是否拥有相应权限，若拥有权限，则允许函数调用；否则通过返回 NULL 拒绝函数调用. 本文结合每个函数的具体特征，对其进行相应的监测代码注入. 下面简述 Landroid/location/Location Manager；－＞getLast-KnownLocation（定位）和 Landroid/hardware/Camera；－＞open（拍照）的监测代码注入过程.

1） Landroid/location/Location Manager；－＞getLastKnownLocation 监测代码注入

由于手机有多种定位方式，每种定位方式都有自己的调用方式；要拦截所有的方式并进行监测代码注入是一件比较麻烦的事情，本文采用的策略是只拦截 GPS 定位并注入监测代码，而其他的定位方式则是拦截后将返回值置 NULL，这样就强迫应用只能使用 GPS 进行定位. Android 源码中有一个叫作 LocationManager 的类，它里面用到关系定位的函数：getLastKnownLocation 和 getBestProvider，以及 requestLocationUpdate 等函数. 本文就将 Landroid/location/LocationManager；－＞getLastKnownLocation 捕获并注入监测代码. 未注入代码前原程序的调用流程语句为：

invoke-virtual {p0, p1}, Landroid/location/LocationManager; －＞ getLastKnownLocation (Ljava/lang/String;)Landroid/location/Location.

Virtual 表明调用的是实例的虚函数，p0 代表 Landroid/location/LocationManager 这一实例对象，p1 是函数的参数.

经过监测代码注入后，原程序的调用流程变为：

invoke-static {p0, p1}, Lrepackaging/ACCESS_FINE_LOCATION; －＞getLastKnownLocation (Ljava/lang/String;) Landroid/location/Location;

用 static 替代 virtual，是为了减少参数的个数，相对于 virtual，静态方法 static 的参数无需包含当前实例对象. 这样参数 p0 就用于表示原 Landroid/location/LocationManager 实例对象的引用，应用在经过监测代码后还能顺利地调用原函数 Landroid/location/LocationManager; －＞ getLastKnownLocation 的功能.

2） Landroid/hardware/Camera；－＞open 监测代码注入

Android 都是基于 C/S 层架构的，虽然具体的工作都是服务端完成，但是客户端上拥有相应的调用接口，便于调用服务端的具体工作，Camera 功能实现方式也不例外. Camera. cpp 客户端接口实现 frameworks\base\libs\camera 拍照功能，Android 通过 JNI 接口将底层的功能传递给 Java 层的 android. hardware. Camera；并且通过 Landroid/hardware/Camera；－＞open 开启拍照功能. 本文就是通过对此函数注入监测代码并封装，来实现对拍照权限的监测与控制. 未注入代码前原程序的调用流程语句为：

invoke-static { }, Landroid/hardware/Camera; －＞open()Landroid/hardware/Camera.

经过监测代码注入后，原程序的流程变为：

invoke-static { }, Lrepackaging/CAMERA; －＞open()Landroid/hardware/Camera.

由于原 Landroid/hardware/Camera; －＞open 并没有输入参数，所以注入监测代码后的函数也没有相应的输入参数.

3） 日志记录

用户初次安装某应用时，不清楚应该授予它哪些功能. 于是监测代码中应该拥有日志记录功能，以便于用户根据行为日志判断该应用是否应该拥有某项权限. 还是以 Landroid/location/Location Manager；－＞getLastKnownLocation 的注入为例，在调用函数之前加入如下代码：

const-string v4, "ACCESS_FINE_LOCATION";

const-string v5, "try to getLastKnownLocation";

invoke-static {v4, v5}, Landroid/util/Log; －＞d(Ljava/lang/String;Ljava/lang/String;)I.

假设应用要访问网络，如 www. baidu. com；相应函数前添加的日志记录代码如下：

const-string v3, "InternetPolicy";

const-string v4, "try to connect internet";

invoke-static {v3, v4}, Landroid/util/Lon; －＞d(Ljava/lang/String;Ljava/lang/String;)I.

2 系统测试

为了测试本方案的稳定性和有效性，实验分为3部分：首先对重打包前后应用大小进行对比，然后在开发的手机客户端(security gurad)上测试本方案的实用性，最后通过对大量样本的重打包检测分析成功率.

2.1 测试环境

本测试环境是在 Windows 7 操作系统下，采用 OPPO R6007 Android 4.4.2 真机和基于 Google 原生态的 Android 4.0.3 模拟器；而测试应用则是在 Google Play 下载的良性应用以及在 VirusShare 申请下载的恶意应用.

2.2 效果评估

本实验对应用进行重打包，通过表3可以看出，打包前后文件大小仅有细微的改变.

表3 重打包前后应用大小对比 KB

应用名称	打包前		打包后	
	APK	DEX	APK	DEX
Skype	40953	8504	41242	8642
Trojan	428	391	431	409
HDPzhibo	2738	1472	2775	1502
Chomp	4083	2502	4153	2584

1) 手机客户端测试

经过静态反汇编分析和监测代码注入后，即可对应用权限进行控制. 图3展现了手机客户端的主界面和敏感权限控制界面.

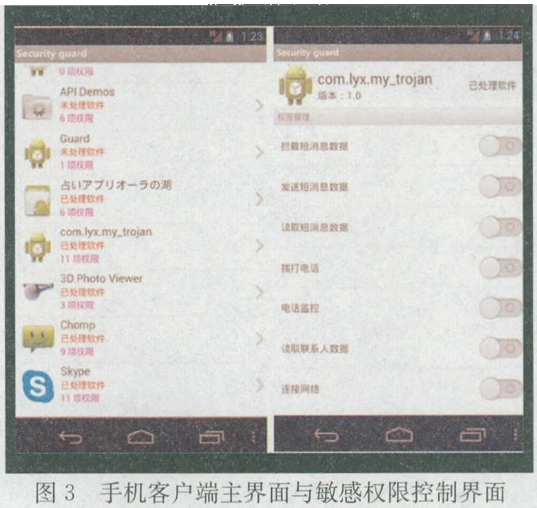

图3 手机客户端主界面与敏感权限控制界面

2) 恶意行为通知和记录

当监测代码监测到可疑的行为时，会即时向用户发出通知，并将可疑行为的详细内容以日志形式进行保存. 同时，用户可以在手机客户端随时查看相应应用的行为日志. 某应用的可疑网络访问行为通知及其日志如图4所示：

图4 可疑行为通知与日志

3) 性能测试

为了测试本文方法的有效性，在 Google Play 官方市场下载包含摄影、视频、购物和通信等124个良性应用，并随机选取其中52个应用；另外收集了包括主流恶意程序在内的48个恶意应用，通过静态反汇编分析、权限分析，对敏感权限成功注入监测代码. 监控代码注入并实现权限控制的成功率统计如表4所示：

表4 样本检测结果

样本类型	样本数	成功注入	成功率/%
良性样本	52	50	96.15
恶意样本	48	42	87.5
合计	100	92	92

根据表4，本实验良性样本数为52个，成功注入50个，正确率为96.15%；恶意样本数为48个，检测出42个，准正确率为87.5%. 计算可得系统的正确率可达92%，所以该系统还是具有可行性和有效性的.

本实验的测试结果未能达到100%，特别是恶意样本成功率有所不足，有如下原因：第一，一些应用进行了代码混淆或加壳，从而造成部分恶意

行为未被检测出来;第二,部分恶意应用会使用相同的 sharedUserId. 这样只需要一个应用申请权限,其他使用相同 sharedUserId 都能使用这些权限,也就逃避了本文的静态分析.

3 结　　语

本文在静态权限分析的基础上,通过在恶意应用权限映射函数中注入监测代码,实现对目标应用的敏感权限实时控制,从而优化了 Android 权限"All-Or-None"的应用授权模式. 初步测试表明,该方案能有效地完成对敏感权限的监控.

参 考 文 献

[1] Fu Yiang , Zhou Danping. Android's security mechanism analysis [J]. Netinfo Security, 2011 (9): 23-25

[2] Wang Shifa, Gao Xianqiang, Han Lu. Analysis of the security mechanism of Android and countermeasures [J]. Electronic Test, 2013 (22): 59-60

[3] Zhu Jiao, Li Hongwei, Peng Xin, et al. On relationship of functions and permissions in Android applications [J]. Computer Applications & Software, 2014, 31(10): 27-33

[4] Lee H-T, Kim D, Park M, et al. Protecting data on android platform against privilege escalation attack [J]. International Journal of Computer Mathematics, 2016, 93 (2): 401-414

[5] Cai Luocheng. Discussion on mechanism for backstage monitors of Android [J]. Information Security and Communications Privacy, 2010 (6): 39-41

[6] Enck W, Gilbert P, Han S, et al. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones [J]. Communications of the ACM, 2010, 57(3): 393-407

[7] Yang Jing, Jin Weixin, Wu Zuoshun. Research on the scheme of permission management and optimization based on Android system [J]. Electronics Quality, 2015 (3): 4-8

[8] Qiu Lingzhi, Zhang Zixiong, Sun Guozi. InsightDroid—Dynamic method trace in Android applications [J]. Journal of Chinese Computer Systems, 2014, 35(11): 2482-2487

[9] Bartel A, Klein J, Monperrus M, et al. Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing Android [J]. IEEE Trans on Software Engineering, 2014, 40(6): 617-632

[10] Chen Weihe, Qiu Daolong. An enhanced Android security mechanism model [J]. Wireless Communication Technology, 2014, 23(1): 37-41

[11] Enck W, Octeau D, Mcdaniel P, et al. A study of android application security [C] //Proc of USENIX Conf on Security. Berkeley: USENIX Association, 2011: 1175-1175

[12] Enck W, Ongtang M, Mcdaniel P. On lightweight mobile phone application certification [C] //Proc of ACM Conf on Computer and Communications Security. New York: ACM, 2009: 235-245

[13] Zhou Yajin, Jiang Xuxiao. Dissecting Android malware: Characterization and evolution [J]. Institute of Electrical and Electronics Engineers Inc, 2012, 4(3): 95-109

雷　磊
硕士研究生,主要研究方向为恶意代码检测.
jellyleiscu@163.com

胡　勇
副教授,主要研究方向为信息安全.
80512430@qq.com