



# 移动互联网时代终端安全问题及解决方案分析

严丽云<sup>1</sup>, 杨新章<sup>1</sup>, 陆 钢<sup>1</sup>, 侯韶新<sup>2</sup>

(1. 中国电信股份有限公司广东研究院 广州 510630; 2. 中国移动集团南方基地 广州 510640)

**摘 要:** 移动互联网端到端的安全涉及云、管、端 3 个维度, 着眼于移动智能终端的安全, 以 Android、iOS 移动智能终端为代表, 从系统级、应用级、漏洞及恶意代码多个角度, 讨论移动互联网时代终端安全问题与解决方案。

**关键词:** 移动智能终端; 系统安全; 应用安全

**doi:** 10.3969/j.issn.1000-0801.2014.12.021

## Analysis of Terminal Security Issue and Solution in Mobile Internet Era

Yan Liyun<sup>1</sup>, Yang Xinzhang<sup>1</sup>, Lu Gang<sup>1</sup>, Hou Shaoxin<sup>2</sup>

(1. Guangdong Research Institute of China Telecom Co., Ltd., Guangzhou 510630, China;

2. China Mobile Group Southern Base, Guangzhou 510640, China)

**Abstract:** Currently, security has been regarded as one of the greatest problems in the development of mobile Internet, the end-to-end security of mobile internet involves three dimensions: cloud security, pipe security, terminal security. The mobile intelligent terminal security, Android and iOS mobile intelligent terminal were as the representative, the security situation on mobile intelligent terminal and their solutions were discussed from the system, application, vulnerabilities and malicious code multi-angle.

**Key words:** mobile intelligent terminal, system security, application security

### 1 引言

当前, 移动互联网在全球掀起了新的发展高潮, 移动智能终端成为信息获取和传播的重要平台。一方面, 人们对移动智能终端的依赖越来越大, 截至 2013 年 6 月, 全球激活的移动智能终端已达 10.89 亿, 其中, 中国激活的移动智能终端数占 24%, 数十亿人每天随身携带存放着他们个人信息的这些小终端, 这些终端的安全问题很重要; 另一方面, 手机用户每天花费在移动应用上的时间也越来越多, 移动应用是最大的受益者, 从而使移动应用和服务不断丰富。截至 2014 年 4 月, 苹果公司 App Store 及 Google Play 的官方应用数量已超过百万款, 第三方非官方 Android 应用商店应用数量也超出了百万款, 这些海量应

用的安全性都是值得深思的问题。

### 2 终端安全问题探讨

#### 2.1 系统安全现状

##### 2.1.1 系统管理机制

iOS 是一个较封闭的操作系统, 苹果公司限制开放其平台上的系统级服务, 并加以严格的管控, 从而减少受攻击面, 目前针对 iOS 的恶意软件数量比针对 Android 系统的恶意软件少很多。

Android 系统是一个半开源的操作系统, 即根据 ASOP (Android 源代码开放计划), 任何人都可以看到其内部操作系统的核心代码, 第三方终端厂商可调用操作系统底层的服务, 定制成有别于标准系统的系统实例, 第三方为保



证自己的利益,定制代码通常是不开源的。半开源系统模式是一把双刃剑:一方面,它方便了第三方集成,有利于 Android 产业链迅速壮大;另一方面,导致 Android 平台凌乱的分辨率和版本分布,在 Android 阵营有超过数百种产品供消费者选择,系统碎片化与更新过期的问题严重,最新版本的市场占有率极低,并且厂商技术能力参差不齐,这些都给黑客可乘之机。

### 2.1.2 越狱风险

大量的用户越狱行为破坏了 iOS 的安全机制,成为 iOS 最大的安全威胁。据统计,全球约有 10% 的 iOS 设备进行了越狱,而这一比例在中国高达 35%。越狱是获取设备 root 权限的技术手段,设备越狱后,对用户而言可以通过 Cydia(越狱软件商店,设备越狱的标志)或自行安装应用,免费使用一些经过破解的收费应用;越狱对开发者而言无需受限于 App Store 严格的审核要求,获得了 root 权限后可以访问系统资源、使用私有 API(application programming interface,应用程序编程接口)、修改系统特性等,可实现拦截短信/通话/邮件、GPS 后台跟踪、破解网络密码、后台录音等,给设备造成极大的安全隐患。

### 2.1.3 刷机风险

刷机是指向设备写入新的固件,即新的 ROM,也称为刷 ROM,用户在对手机解锁后,可以自行更换或者定义设备原有系统固件。目前针对 Android 系统进行刷机的行为较为常见,而针对 iOS 的较少,原因在于 Android 系统在升级时,除了 Google 推出的 Nexus 系列手机可以获得官方升级外,其他系列手机往往需要借助刷机获得升级。这种刷机行为存在风险,一些不法业者可以在手机的 ROM 做手脚,让用户束手无策。

## 2.2 应用安全现状

### 2.2.1 iOS 应用安全现状

苹果公司较为关注其平台生态系统的健康状况,它对所有在其平台上发布的应用有一套较为严格的认证及审核制度,iOS 的应用必须得到受信任一方(如苹果公司)的签名,否则 iOS 中的强制性代码签名(mandatory code-signing)需求会让这些应用无法在设备上运行。另外,来源于 App Store 的应用会以较低级别的权限运行在沙盒中,这种方式可以降低它们的破坏性,在一定程度上监管了 iOS 应用级的安全。

### 2.2.2 Android 应用安全现状

Google Play 是 Android 官方应用市场,相比 App Store

严格的审核机制,Google Play 的门槛宽松得多,采用先上线再审查的审核政策,并没有对应用进行严格审查和筛选。

此外,第三方应用商店能够提供全面的、跨平台的应用软件下载,因此使用率也非常高,用户被第三方应用商店大量分流,国内 95% 的用户都会选择从国内的第三方应用市场下载应用,而第三方应用市场存在市场混乱、分散现象,成为恶意软件滋生的土壤,威胁着用户的利益,同时也给开发者带来极大的困扰,他们需要“伺候”太多的第三方渠道以增加应用的曝光率;在应用管理方面缺乏应用监管,安全检测缺乏权威性,经过较知名的第三方应用市场的应用发布测试发现,发布应用存在的漏洞、后台偷跑流量、窃取用户通讯录等问题均可在被测应用市场中审核通过,应用市场的审核形同虚设,大量的盗版应用出现在第三方应用市场中;在应用数量和质量方面,虽然第三方应用市场应用数量庞大,但存在应用价值低的现象,在 17 个 Android 应用商店中对有下载量数据的 33 万多个应用程序进行统计,下载量超过 1 000 万的应用程序仅有 80 个(去重),绝大部分应用程序的下载量少于 1 万次。

趋势科技发布的《2013 年第三季度信息安全总评季报》显示:截至 2013 年 9 月,针对 Android 平台的高危险恶意程序数量达到 1 005 000 个,而且以付费服务、盗用软件为主,这类程序盗取手机用户的账号权限,在用户不知情的情况下发送短信或申请付费服务,其余 20% 为包括越权广告程序在内的高风险应用程序。除了应用市场混乱外,基于 Android 软件的封装形式,使得黑客很容易对流行软件代码进行反编译,植入恶意代码,再将软件散布在第三方应用市场,严重侵害了用户和开发者的权益。

## 2.3 漏洞及恶意代码猖獗

漏洞和恶意代码是安全领域邪恶的一对,系统越狱、刷机等行为及应用市场乱象会直接导致漏洞的滋生及恶意代码的注入。漏洞通常是开发者无意的过失,而恶意代码则是黑客有意的行为,黑客利用了设备中系统底层或软件漏洞运行其恶意代码,用户甚至不需要参与,恶意代码就会利用某个漏洞在设备上运行。

漏洞分为系统漏洞和第三方漏洞,系统漏洞主要包括:任意代码执行、签名漏洞、提权漏洞、内存信息泄漏、数据信息泄漏等。第三方漏洞主要包括:Web 相关漏洞、信息泄漏、自带库导致的任意代码执行以及逻辑错误等。

随着智能终端的普及,恶意代码的数量增加很快,恶意代码和手机代码对抗形式从围剿到被围剿,某类恶意代

码的发展演变阶段介绍如下。

- 阶段一(代码形式):恶意代码段捆绑到正常应用代码中。
- 阶段二(DLL 文件形式):恶意代码控制部分演变为 DLL 可执行文件,分析难度提高。
- 阶段三(混合件形式):包装成第三方提供的混合件捆绑到正常应用中。
- 阶段四(恶意代码加密):对恶意程序隐藏、混淆、加密,提高抗分析力。

恶意代码整体呈现出变种多、变化快的特点。据统计,恶意代码平均一个季度超过 6 次变化,侵入层次也从 Android 应用级别发展到 Android 文件、Android 系统级别,跨平台恶意代码、混合应用恶意代码也逐渐成为趋势,这些都加剧了恶意代码被查杀的难度。

整体安全解决方案如图 1 所示,从应用运行前在应用市场中的安全机制到应用运行时在系统中的安全保障,再到开发者及研究人员对应用漏洞及恶意代码检测这 3 个维度进行终端安全解决方案剖析。

### 3 应用市场措施研究

根据应用存在的形态,可将应用级安全分为应用安装前、应用安装后的安全保护。应用安装后主要依靠系统级别的安全措施,而应用安装前主要存在于应用市场云端,恶意软件排除应从源头抓起,下面将探讨几种应用市场安全措施。

#### 3.1 应用市场的签名机制

应用市场的签名机制在系统 iOS、Android 系统中的用途各有不同。在 iOS 中要求所有二进制文件和类库在被内核允许执行之前,都必须经过受信任机构(如苹果公司)的签名,是 iOS 中最重要的安全机制,所有的应用都必须从苹果公司的 App Store 下载(除非对设备进行重配置、越狱

等)才允许在设备上运行。在 Android 系统中签名机制主要用于程序之间建立信任机制和区分相同类名、安装包名。签名可以保证相同名字但签名不同的安装包不被替换,禁止更新、安装签名不一致的 APK(Android 安装包),从而防止已安装的应用被恶意的第三方覆盖或替换;签名还可以让同一签名不同名字的安装包程序运行在同一进程中,从而允许彼此访问对方的代码和数据,实现代码、数据共享。与 iOS 不同的是,Android 系统用于签名的数字证书是由非权威机构认证,应用签名可由第三方完成,如应用程序商店、运营商或 OEM 厂商,也可以由开发者完成自签名。

#### 3.2 应用市场的应用包加密

Java 语言开发的 Android 应用非常容易被破解和篡改,从而对应用进行反编译、二次打包、恶意代码或广告注入等不良行为,对应用包加密正是针对这些行为的解决方法,其主要目的是防逆向工程、防篡改、防复制。目前 Google Play 已具有对 Android 应用安装包的加密功能,能在 Android 4.1 Jelly Bean 系统实现将应用与设备绑定,使其更难以被破解和无限制地分享,可以一定程度地保护应用的安全。国内也有许多第三方机构在应用上传市场前,对应用进行加密,它除了对应用安装包实现了加密之外,还有其他安全检测工具,如防非法注入检测、防二次打包检测等。

#### 3.3 云端检测工具

运用恶意代码检测工具实现应用市场云端对恶意软件进行排查,云端一旦加载一个应用程序,这项服务立即开始分析这个应用程序,查找已知的恶意软件、间谍软件和木马程序,如谷歌“保镖”就是谷歌推出的基于此原理设计的云端检测工具。

#### 3.4 第三方应用市场监管

Android 系统的开放性决定了其应用市场不可能唯一,第三方应用市场大量存在使得应用发布的源头错综复杂,给黑客可乘之机。在互联网经济环境下,需要建立健全

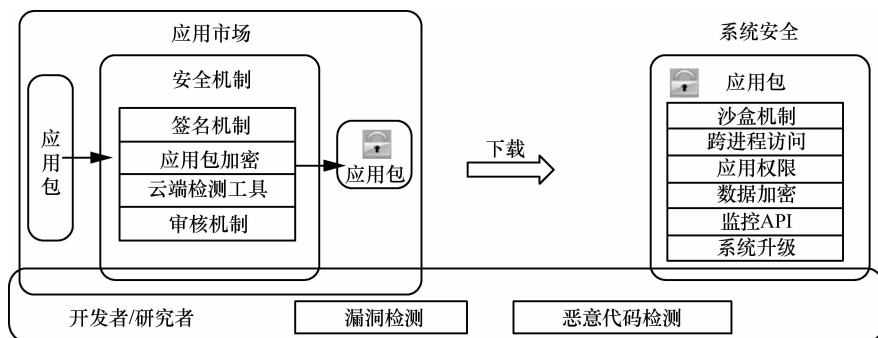


图 1 整体安全解决方案



的相关法律法规对互联网相关产物进行法律界定,就应用市场责任而言,应当按照实际情况加以判断。对于有盈利性质的或对上传的应用进行审核、推荐、编辑的应用市场,对应用安全性应该承担相关法律责任,否则无须承担法律责任,但有责任告知用户。目前国内第三方应用市场多数是盈利的,应在此范围内。为实现应用市场责任的可溯性,应用市场应该对平台上的应用打上唯一标识。

### 3.5 第三方应用市场审核机制建议

建议从开发者管理、检测技术、人工审核、审核流程、行业标准及版本更新管理等多个维度改进第三方应用市场的审核机制。开发者管理方面需建立开发者实名认证机制及审核机制,开发者需对应用负责,开发者责任可溯,从而遏制盗版应用和恶意软件,也有助于规范行业;安全检测技术方面需提高技术手段,以开放合作的方式引进业界安全服务提供商相关技术,既要有针对病毒特征库的检测机制,又要基于静态数据和行为数据分析对新病毒(0daymalware)进行检测;审核方面需建立完善的审核流程及人工审核团队,审核流程需明确定义被拒绝应用的特征,作为人工审核依据;协助建立标准化的行业规定,如对应应用权限使用建立明确的行业规定,作为人工筛查的重要依据,改变原有应用权限,使用完全交由用户决策的机制;对应应用版本更新管理方面,更新应用仍需要进行安全检测。

## 4 系统级的安全方案

系统的安全决定了上层应用及用户数据的安全,Android 系统、iOS 自带了许多安全机制,从权限、进程、接口、数据、安全环境不同层面保护了应用运行时的安全及数据安全。

### 4.1 沙盒机制

系统中应用程序有自己的文件系统,且只能在该文件系统中读取文件,外界也不可以访问该文件系统区域,此区域被称为沙盒。所有的非代码文件都要保存在沙盒内,如图像、声音、文本文件等。Android 系统、iOS 通过沙盒实现应用程序之间的分离和权限控制,这里权限指访问系统和其他应用资源的权利,如文件和目录、传感器和网络。

Android 系统是基于 Linux 内核设计的,它的沙盒机制是基于 Linux 用户 ID 权限控制实现的,应用在安装时会被分配一个固定的用户 ID,这个用户 ID 与内核 Linux 系统用户 ID 对应,使得应用运行于不同 Linux 用户进程中,从而实现数据隔离。相同的用户 ID 用户资源可以互相访

问,可以通过共享用户 ID 的方式实现对系统资源或其他资源的访问,需要应用在 manifest 文件中使用相同的用户 ID。

iOS 的沙盒是基于 TrustBSD 策略框架的内核扩展模块,应用运行在不同的进程中,针对每个进程都可以制定特殊的沙盒配置文件,提供细粒度的应用权限访问控制,沙盒配置文件以正则表达式的语法来描述,在应用运行时启动沙盒,沙盒的配置会被传到系统内核中执行,从而对每个进程的行为进行访问控制。

Android 系统、iOS 访问控制的共同点在于二者均是基于某种程度的强制访问控制(mandatory access control, MAC)模型技术实现的,保护应用程序存储区域的安全,不同的是 Android 系统是由操作系统之上的中间层实现 MAC 技术,iOS 是操作系统层面的 MAC 技术。MAC 技术主要特征是对所有主体(可以授予或拒绝访问某个对象的人或物)及其所控制的客体(如进程、文件、设备)实施强制访问控制。为这些主体及客体指定敏感标记,这些标记是等级分类和非等级类别的组合,它们是实施强制访问控制的依据。系统通过比较主体和客体的敏感标记来决定一个主体是否能够访问某个客体,用户的程序不能改变自身及任何其他客体的敏感标记。在主体方面二者也存在差异,Android 系统沙盒机制要求用户对应用进行授权,由用户负责安全监管,而 iOS 沙盒机制是基于应用商店实现苹果公司严格的应用审核,实现应用监管的,弥补了用户的技术缺陷。

### 4.2 跨进程间通信安全机制

#### 4.2.1 Android 系统跨进程间通信机制

Android 系统沙盒机制决定了应用进程间的隔离,而在 Android 系统中应用的跨进程协作无处不在,如游戏类应用需要调用社交类应用接口进行成绩分享等。Android 系统通过组件实现跨进程的通信,组件是 Android 系统应用安全管理的最小粒度的单元,掌握组件间通信的原理是理解 Android 系统安全模型的基础。组件必须在 Android 系统应用的配置文件 Androidmanifest.xml 中定义,由系统管理进行调度,具有独特的生命状态周期。主要涉及的组件有活动、服务、内容提供者、广播接收者等,活动组件是负责人机交互的组件,一个界面可以看作一个活动组件;服务组件是负责在后台长期执行的服务,如播放器应用在界面推出后,其服务仍然在后台运行,这是靠服务组件实现的;内容提供者组件用于管理应用中可共享的数据,可以

实现应用间数据的共享;广播接收者组件用于接收系统范围内的事件声明,如电池电量不足告警等。应用、进程及组件的关系如图 2 所示。

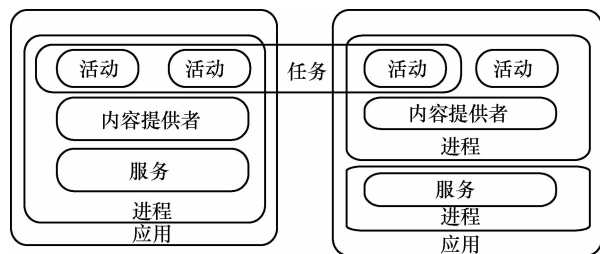


图 2 Android 系统应用、进程及组件关系

跨进程间通信的机制有 Intent、IPC binding 及 content resolver,Intent 实现活动组件、服务组件、广播接收者组件唤醒;IPC binding 机制是使用远程函数调用的方法与服务组件通信;content resolver 用于与内容提供者组件通信。

然而,黑客针对这些进程通信机制制造出了许多恶意行为,如恶意调用、恶意发送广播等。常见的风险规避措施主要包括两个方面:一是最小化组件暴露,对于应用私有组件,即无须参与进程通信的组件,可将其属性设置为 android:exported="false",因此只有相同用户 ID 的应用才能启动该进程;二是对于需要参与进程间通信的组件、公开的广播组件进行权限设置,声明<permission>属性,设置保护级别,另外在运行时对暴露组件接口进行实时检查也是很有必要的,Android 系统在 android.content.Context 类中提供了运行时检查的各类 API。

4.2.2 iOS 跨进程间通信机制

iOS 对应用间的通信管理非常严格,若要实现 iOS 应用间的协作,需要在.plist 文件中注册对外接口,在调用的应用中添加代码,将接口 URL 转化成 NSURL 的对象,然后将它传给共享 UIApplication 对象 openURL。openURL 方法会启动注册接收该 URL 类型的应用程序,并将 URL 传给该应用程序,被调用的应用程序需要实现 application:handleOpenURL 方法,从而处理传递给它的 URL 请求,实

现应用进程间的协作。

4.3 应用权限

应用程序之间的不信任关系是通过沙盒机制隔离的,而应用与系统 API、资源之间的访问控制是通过应用权限控制实现的,默认情况下应用程序不具备访问系统资源的任何权限,需要进行相应的权限申请。

Android 应用权限基于 permission 机制。获得系统访问权限需要在应用程序 manifest 文件中通过制定<permission>、<permission-group>、<protectionLevel>等,声明需要访问的 API 及资源,见表 1。在应用安装时需要用户授权,在应用运行时需要检查、执行、授予和撤销的权限。

iOS 应用权限是开发者在程序设计中选择开启的,开启的位置位于工程文件下 target 栏目下的 Capabilities 选项卡,其中列出了可被调用的 iOS 设备能力,在 Info.plist 文件中将列出声明的设备能力,如 iCloud、SMS,如图 3 所示。在应用上传给 App Store 审批的过程中,如果使用了非公开的 API,该应用将被拒绝发布;在用户通过 App Store 下载相关应用前,App Store 会利用 Info.plist 文件中的声明检查设备能力,以避免用户下载不符合自身设备能力的

4.4 监控 API

恶意应用行为主要表现形式是在智能终端后台未经许可地调用敏感 API,其不易被用户察觉,如监听信息数据库、监听联系人数据库、私自录制视频、私自拍照等行为,对用户财产安全和信息安全构成重大威胁。可以对语音通信、通信记录、图像获取、GPS 位置获取、蓝牙及数据传输等敏感 API 进行监控,阻止恶意应用行为的侵害。技术实现上,一种可行的方式是通过安全防病毒软件实现对系统级 API 的监控,如 360 手机助手、金山手机卫士等。这种方式是在系统未集成权限管理功能之前常采用的安全手段,一定程度上实现了对应用的监控,而安全防病毒软件需要取得操作系统内核级的较高权限,才能对系统级 API 进行监控,显得美中不足;随着系统升级及用户对权限管理的需求日趋强烈,iOS 及 Android 系统分别集成了

表 1 Android 的权限标签

| 声明权限标签             | 含义                                                                                                                    |
|--------------------|-----------------------------------------------------------------------------------------------------------------------|
| <permission>       | 权限名称                                                                                                                  |
| <permission-group> | 权限组,用于定义一组权限,以某种规则划分权限集合                                                                                              |
| <protectionLevel>  | normal 级别只要是申请的权限就可以用<br>dangerous 权限在安装时需要征得用户确认才可用<br>signature 权限与 signatureorsystem 权限需要应用是系统用户,如系统制造商,属于系统程序可用权限 |

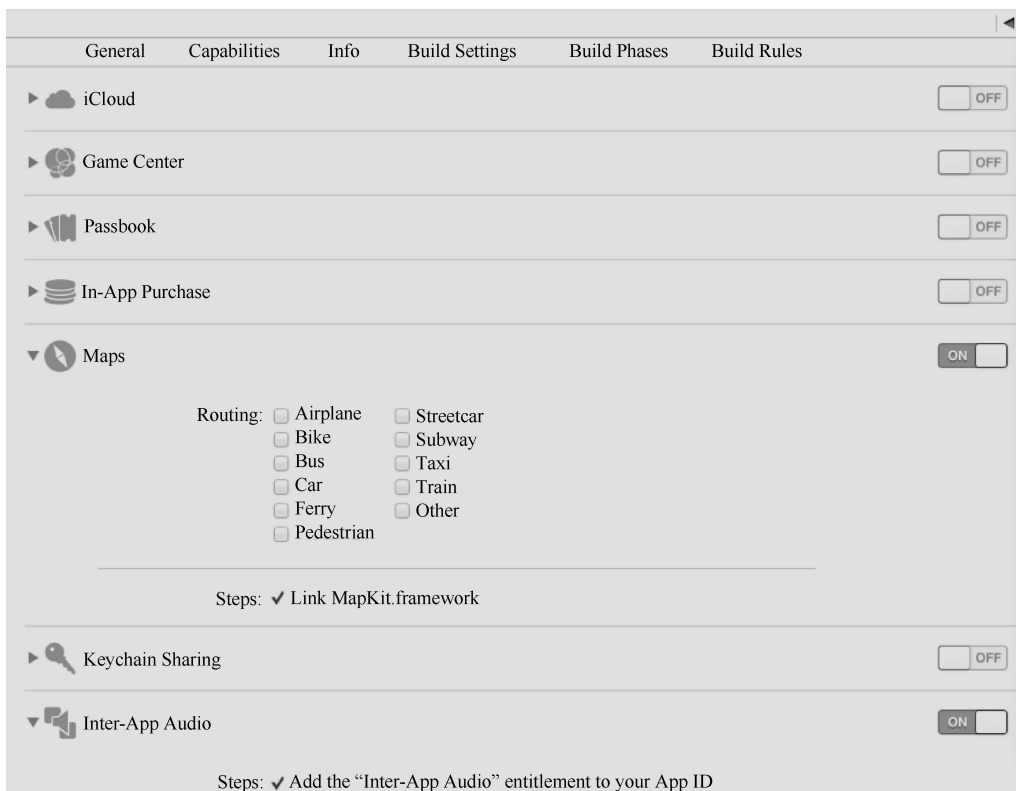


图3 iOS 设备能力

权限控制功能,无须用户对设备进行 root,即对所有应用建立访问控制列表,对于被调用的敏感 API 进行动态授权控制,依据访问控制列表限制应用行为。目前在 iOS6 操作系统中系统集成了这项功能,通过“设置”→“隐私”窗口进行权限项控制,用户可以针对某一应用选择开启或关闭权限



图4 iOS 监控 API

项,如图4所示。从 Android4.3 开始也新增了此安全机制,在设置中开放 AppOps 功能,对应用设置访问位置、信息、设备等相应 API 的控制列表,如图5所示。

#### 4.5 数据加密

数据库存储是经常用到的一种存储方式,Android 系

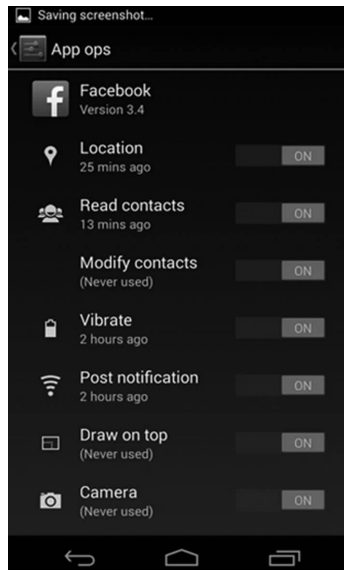


图5 Android 监控 API

统内置了 SQLite 数据库,实现对数据的存储和操作。当 Android 手机获得 root 权限后,即可访问/data/data/<package\_name>/databases 下存储的所有数据,如果数据涉及用户隐私或账号安全等,则存在严重的数据安全隐患,Android 系统中提供 SQLCipher 来解决这一安全问题。SQLCipher 是一个在 SQLite 基础之上进行扩展的开源数据库,它主要是在 SQLite 的基础上增加了数据加密功能,使用 AES 加密算法实现,如果在项目中使用它来存储数据,就可以大大提高数据的安全性。

iOS 使用硬件密钥保护文件系统中的系统和数据分区,系统和数据分区中的数据将由一个基于硬件设备的密钥进行加密,该密钥存储在一个 AES 加密加速硬件上。这个基于硬件的密钥不能被 CPU 访问,只有在加解密时才能由该加速器获取。通过这种机制,直接从硬盘上被取走的原始数据就无法被正确解密。同时,数据加密机制还提供了多种加密策略供应用程序选择。应用程序可以选择对一些敏感文件使用 NSFileProtectionComplete 策略而不是默认的 NSFileProtectionNone 策略加密数据,前者使用用户口令和硬件密钥结合的方式加密数据,在设备解锁后才能读取;而后者只是用硬件密钥加密数据,在需要时可随时解密,相对于前者来说,对数据安全更宽松。

#### 4.6 系统升级

系统升级不仅是系统功能更新的重要手段,同样也是系统不断增加其安全性能的重要途径,系统升级都会修复上一版本中存在的许多错误。

iOS 在升级过程需要联网进行验证,系统升级前设备会将 LLB、iBoot、内核、镜像和一个随机的且不可重复的值发送到苹果的服务器进行验证,其中 LLB、iBoot 用于系统启动中的验证,可以保证 iOS 只在苹果设备中才能启动。系统升级前服务器端对上述参数进行验证,如果通过验证,将会返回一个通过的结果,结果附加了与设备唯一相关的 ECID,此值只能对应一台设备,且只能使用一次。通过这种机制,保证了系统升级过程都是符合苹果要求的,提高了安全性。iOS 升级之后是不允许降级的(官方没有提供接口),因此系统的安全等级只会越来越高,而不会出现由于系统降级,已修复安全风险又暴露出来的情况。

Android 系统升级是由手机厂商控制的,并且系统可以降级。除了 Google 自己出的 Nexus 系列手机可以

获得官方升级外,其他系列手机都是由手机厂商提供升级包,由用户自行选择、下载相应升级包进行系统升级,而手机厂商无法顾及整个生态链,因此 Android 系统升级周期存在时延问题,成为威胁系统安全的一大症结。

## 5 防漏洞攻击和恶意代码检测

除了从系统及应用层面加强防御手段外,还应从安全问题的源头(即漏洞及恶意代码)出发遏制隐患。

### 5.1 防漏洞机制

一方面,借助漏洞查找工具尽可能地排除已知漏洞。由于发现一个未知漏洞比发现一个已知漏洞困难得多,因此多数攻击者利用的是已知漏洞,漏洞查找工具依据这个原理,建立漏洞利用程序数据库,漏洞利用程序样本更新的效率通常是考察一个漏洞查找工具优劣性的重要指标,较为流行的漏洞查找工具有 Canvas、Core Impact 和 Metasploit Framework,其中 Metasploit Framework 是测试较为常见的缓冲区溢出漏洞的辅助工具。

另一方面,加大漏洞被发现的难度,保证最小权限。黑客是从系统和软件可以被外界访问的那部分代码中发现漏洞的,但系统和软件总是需要与外界交流的,这就不免向外暴露一些代码,应尽可能少地将代码暴露给攻击者,建议只提供必要的 API。

### 5.2 恶意代码的检测

常见的恶意代码检测方法可分为:基于病毒特征签名、基于行为和启发式的检测。基于病毒特征签名的原理是指通过对已知恶意软件样本使用签名的方式来识别恶意软件,然而该机制下对应用软件病毒的探测程度取决于数据库对病毒样本库的积累,黑客们通常会不断重写恶意代码,使检测工具无法识别。病毒样本库需要维护每个变种病毒的特征签名,数量不断膨胀,增加了病毒检测的复杂度和能耗,并且该机制无法识别新的病毒(0day malware),于是就出现了基于行为的检测方法,它是通过分析应用代码寻找其中潜在安全威胁的恶意指令来实现检测的,如利用系统漏洞获得 root 权限、私自发短信等恶意指令,RiskRanker 自动安全监测系统正是基于这一策略设计的一款反病毒软件。启发式检测方法原理是指基于恶意代码调用的系统 API 所处的危险级别实现检测,通过静态和动态分析获得恶意代码调用的函数列表和函数间的调用关系,经过大量样本分析,总结出特征规则并对应每种调用关系定义威胁



级别,如果某个程序调用了威胁的特定函数集合,则可以怀疑其为恶意代码,这种方式较容易出现错判。当前单一的检测技术无法完成对日益变化的恶意代码排查,应该综合多种检测技术并结合人工检测才能较好地封堵恶意代码。

## 6 结束语

移动智能终端成为人们的第一终端,是利益与威胁的焦点,其安全性必须由产业链条上下游的各方共同保障,通过开放合作构建云、管、端安全的产业闭环。云端应用市场应加强审核机制和检测手段,以提供安全的移动应用平台,云服务提供商需加强对分布式架构下安全的研究,以提供安全的云服务;运营商智能管道作为用户信息的载体,安全服务是智能管道中应该加强的方面,需应对管道扁平化及承载 IP 化带来的新的安全问题,除提供硬件安全外,还应该开放合作为用户提供安全服务;终端制造商及系统提供商需应对开放带来的安全问题,终端厂商推出的新产品应搭载最新版的系统,并确保旧产品的快速系统升级,以阻止系统漏洞横行。

## 参考文献

- 1 McGraw G, Morisett G. Attacking malicious code. A Report to the Infosec Research Council, 2000(5): 33~41
- 2 Rafael F, Christian B, Christoph K, *et al.* Android OS Security: Risks and Limitations. AISEC Technical Reports, AISEC-TR-2012-001, 2012

(上接第 144 页)

- 27(10A): 98~101
- 5 罗志强,史国水,沈军等.移动互联网安全热点技术研究.电信科学,2013,29(Z1): 254~256
- 6 袁春阳.移动智能终端应用软件安全检测.信息安全与通信保密,2012(10)

### [作者简介]



薛立宏,男,中国电信股份有限公司上海研究院高级工程师,主要研究方向为移动互联网应用及信息安全。

- 3 罗喧,梁柏青等.智能手机应用安全关键技术探讨.电信科学,2013,29(5): 19~27
- 4 许方恒,陈喧,沈才樑等.移动互联网核心安全需求及防护研究.电信科学,2013,29(11): 77~81
- 5 岳荣,李洪.探讨移动互联网安全风险及端到端的业务安全评估.电信科学,2013,29(8): 74~79
- 6 房秉毅,张云勇,吴俊等.云计算应用模式下移动互联网安全问题浅析.电信科学,2013,29(3): 41~47
- 7 Nguyen-Thanh N, Koo I. Empirical distribution-based event detection in wireless sensor networks: an approach based on evidence theory. IEEE Sensor Journal, 2012, 12(6): 2222~2228

### [作者简介]



严丽云,女,中国电信股份有限公司广东研究院工程师,主要从事云计算虚拟化技术研究及移动互联网产品研发工作。

杨新章,男,博士,中国电信股份有限公司广东研究院高级工程师,主要从事通信技术与产品研发工作。

陆钢,男,中国电信股份有限公司广东研究院高级工程师,主要从事移动互联网、云计算技术研究工作。

侯韶新,男,中国移动集团南方基地工程师,主要从事云计算、IDC 技术研究工作。

(收稿日期:2014-07-28)



郑京芳,女,中国电信集团公司高级工程师,主要研究方向为网络信息安全。

蒋永成,男,中国电信股份有限公司上海研究院助理工程师,主要研究方向为网络信息安全。

王渭清,男,中国电信集团公司工程师,主要研究方向为网络信息安全。

(收稿日期:2014-07-09)