



专题:移动智能终端技术与应用

智能手机应用安全关键技术探讨^{*}

罗 喧,梁柏青,潘军彪,黄 粤

(中国电信股份有限公司广东研究院 广州 510630)

摘 要:首先针对日益严峻的移动终端应用安全问题,深入探讨移动终端操作系统的安全隔离机制,总结 Android、iOS 主流操作系统的应用安全沙盒的实现机制,在此基础上就用户级运行时权限回收、应用组件进程间通信、敏感信息流向跟踪等关键技术进行研究。

关键词:手机安全;沙盒;自主访问控制;强制访问控制;信息跟踪;Android;iOS

doi: 10.3969/j.issn.1000-0801.2013.05.004

Study on the Key Technologies of Smartphone Application Security

Luo Xuan, Liang Baiqing, Pan Junbiao, Huang Yue

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

Abstract: Mobile phone security has become increasingly important in mobile terminal. It is of particular concern as it relates to the security of personal information now stored on smartphones. Some key technologies will be greatly helpful to improve the security level of mobile device, such as the sandbox architecture of the mobile OS, the permission revocation on runtime, the inter-communication between mobile applications and components, and the information-flow tracking. It was aimed to explore and analyze the security key technologies, and showed the future trends of mobile phone security.

Key words: mobile phone security, sandbox, discretionary access control, mandatory access control, information-tracking, Android, iOS

1 引言

随着移动互联网的异军突起,用户人数迅猛增长,作为移动互联网的访问入口,智能手机得到了极大的发展,出货量呈现井喷式增长,根据 IHS iSuppli 公司的研究分析,2012 年我国手机供应商的智能手机出货量预计达到 1.014 亿部,比 2011 年的 5 200 万部劲增 94%,几乎是 2009 年 1 020 万部的 10 倍。智能手机正在成为人们身边最不可或缺的设备,是人们与外部世界联系的入口,与工作、社交、娱乐、生活息息相关。

然而,在人们利用智能手机享受移动互联新生活时,

智能手机的安全却遭遇前所未有的挑战。《网秦 2012 年智能手机安全报告》显示,2012 年上半年累计查杀的手机恶意软件有 17 676 款,直接感染手机 1 283 万部,比 2011 年同期增长 177%。金山网络、TechWeb 以及机锋网联合发布的《2011-2012 年度智能手机安全报告》的数据显示,2011 年智能手机安全威胁急速上升,日均新增手机病毒数量翻 10 倍,病毒相关产业获利超过 3.6 亿元,超过 600 万用户通过软件市场、论坛等渠道中毒。智能手机的安全问题愈演愈烈,正在成为悬挂在人们头顶的利刃,成为移动互联网发展的最大制约因素之一。

业界定义的智能手机是一种底层硬件、核心操作系

^{*} “核心电子器件、高端通用芯片及基础软件产品”国家科技重大专项基金资助项目“移动智能终端操作系统研发”(No.2012ZX01039002-001)



统以及上层多样化应用无缝集成的一体化产品。智能手机应用商城中的应用软件数以亿计,其软件安全问题的根本在于移动操作系统有若干技术问题,其对建立安全的智能手机应用环境起着关键的作用。其中操作系统提供的沙盒(sandbox)技术是智能手机应用安全的基础软件生存环境,提供了应用隔离的机制与资源权限访问的模式;其次,应用运行时权限回收技术为用户提供了任意时候的权限吊销管理;再次,智能手机上的组件安全技术是解决如何拦截恶意软件利用组件间的通信进行非法获取权限的恶意行为的方法;最后,敏感信息的流向跟踪技术,用来研究如何拦截敏感信息被恶意应用非法窃取传输到网络。

2 移动终端操作系统的应⽤安全机制

2.1 沙盒技术概述

在计算机安全中,沙盒是一种将运行中的应用隔离在有限的范围内,防止应用破坏系统以及影响其他应用运行的机制。现代移动终端能运行大量的第三方来源应用,第三方的应用软件很可能夹杂着危害系统及其他应用的行为。因此,移动终端的操作系统均使用某种沙盒技术对应用运行环境隔离,使应用运行在沙盒中,能且仅能访问应用需要的数据和系统资源,降低了应用破坏系统和数据的可能性。沙盒技术是整个移动终端应用安全的基础。对应用而言,沙盒定义了一组系统对象集合,并由系统限制应用能且仅能访问定义的对象。

操作系统的安全访问控制的模型通常表述为一个主体(subject)可以访问哪些对象(object)。主体是指可以授予或拒绝访问某个对象的人或事物,如用户、程序、系统进程;对象通常是指被访问的某种系统的资源,如文件、打印机等。目前操作系统安全隔离技术包括自主访问控制(discretionary access control,DAC)和强制访问控制(mandatory access control,MAC)两种类别,后者是安全的操作系统必须的选择。

DAC被定义为“一种基于主体的身份或者主体所属的组别来限制对象的访问权限”,DAC的主要技术特征是主体所具有的访问权限能够通过继承或者赋予被传递给另外一个主体,这意味着访问权限具有传递链条,因此,当一个程序中发生安全裂缝时,便会危及系统,这使得DAC在木马前特别脆弱。目前,最著名的DAC实现是基于用户ID和组(group)的UNIX/Linux文件系统的权限系统。

举例来说,Linux的文件系统中,某个用户A拥有file1的文件,对file1拥有读写权限,对其他用户关闭了读写权限。一个恶意攻击者用户C,写了一个程序,这个程序在运行时生成文件file2,程序中设置新的访问列表:用户A对file2的写权限和用户C对file2的读权限。用户C把恶意程序伪装成合法程序发给用户A,当程序被A运行时,程序具有了A的访问权限,程序逻辑中拷贝file1到file2,于是用户C窃取了file1的内容。如果用户A是系统管理员,攻击者C会获取最大的权限。

与此对比,在MAC模型里,由管理员制定策略,策略定义了哪个主体能访问哪个对象。MAC对所有主体及其所控制的客体(如文件、设备、系统资源)实施强制访问控制,为这些主体及客体指定敏感标记,作为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。如果没有被管理员显式授权,用程序自身不能改变它自己及任何其他主体、客体的敏感标记,从而增加了安全的防备。

MAC的实现中,有多种对象标记和策略判断规则,不同的MAC系统实现并不一样。

在目前两种主流的操作系统iOS和Android的应用沙盒技术中,均采用了某种程度的MAC的技术实现。前者在操作系统内核层面实现了MAC,而后者在中间层实现了MAC。

2.2 iOS的应用沙盒

苹果公司的iOS的应用沙盒,是一种强限制的结构,根据苹果公司的开发文档描述,iOS的应用沙盒将应用限制的级别定义为“每一个应用都是一个孤岛”。为了软件安全,这种设计极大程度地推崇应用隔离,牺牲了本机内应用间的信息共享。

图1示意的iOS应用沙盒来自苹果公司的官方文档。

应用“孤岛”是如何形成的呢?苹果公司的iOS的应用沙盒提供细粒度的应用权限访问控制。其应用沙盒的主要访问限制可以总结如下:

- 应用只看到沙盒容器目录,下面表述为<Application_Home>,不可见系统的其他目录和整个文件系统,沙箱中关键子目录,如<Application_Home>/AppName.app、<Application_Home>/Documents/、<Application_Home>/Documents/Inbox、<Application_Home>/Library/等,每个子目录的使用方法有严格规定;
- <Application_Home>/Documents/Inbox只有读和删除的权限,没有写的权限;

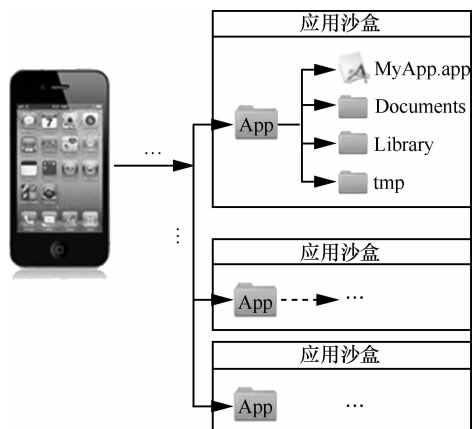


图 1 iOS 的应用沙盒

- 应用可以对用户的照片、视频内容及 iTunes 目录进行只读访问；
- 应用可以对用户的联系人数据 (Sqlite 本地文件数据库) 进行读、写访问；
- 应用可以启动网络连接发送和接收数据；
- 应用仅通过系统 API 和服务的控制执行有限制类型的后台服务；
- 应用不可以连接本地的 UNIX 网络连接字服务；
- 应用不可以读取系统的日志目录；
- 不在权限列表中描述的操作均不能通过授权等。

iOS 的应用沙盒并不是一种基于 UNIX 用户 ID 的权限控制 DAC 方案，而是操作系统内核层次的 MAC 的实现，操作系统集成了 TrustedBSD MAC Framework 项目，实现应用沙盒。

iOS 的应用沙盒执行结果，可以参照 iOS 操作系统的同源操作系统 MAC OS X 的 sandbox-exec 命令来观察，在 MAC OS X 中，存在着使用沙盒机制运行应用的命令：

```
sandbox-exec [-f profile-file] [-n profile-name] [-p profile-string] [-D key=value ...] command [arguments ...]
```

如果在一个没有 Internet 访问权限的沙盒中运行 ping 命令，将直接返回权限不足够。

```
$ sandbox-exec -n no-internet ping www.google.com
PING www.l.google.com (209.85.148.106): 56 data bytes
ping: sendto: Operation not permitted
```

iOS/MAC OS X 对不同的应用类型，定义了不同的沙盒。沙盒的访问权限定义配置文件可以使用 SBPL (sandbox policy language)，以正则表达式的语法来描述。在应用启动的同时沙盒启动，沙盒的配置被传递到操作系统内核执行。

iOS 应用沙盒的强大之处不仅在于单纯的技术实现，还在于苹果公司强大的运营能力。应用在发布到苹果的应用商店之前，经过苹果公司严格的审核测试，如果应用不遵循沙盒的设计规格，将不能正常运作，或者在审核环节被废弃，即使在上市之后如果被发现有恶意的行为仍然会被下架。

2.3 Android 的应用沙盒

Android 操作系统的沙盒技术，是基于 Linux 的原生进程与用户账号组合进行限制的功能。Android 是一种多用户的 Linux 操作系统，每个应用使用不同的用户 ID 运行进程，并对应用的数据文件进行 Linux 操作层次的文件访问保护，赋予且仅赋予程序用户的 ID 以访问权限，使用其他用户 ID 运行的程序无法越权访问程序所保护的数据。

与此相关的另一个技术特点是：Android 应用签名不需要权威的中心进行认证，其验证签名仅是为了区分应用的提供商身份，相同应用提供商签名的多个应用可以运行在同一个进程空间，彼此更紧密耦合地共享数据访问。

操作系统层面基于 Linux 用户 ID 的权限控制是一种 DAC 的权限控制，DAC 的缺点如 2.1 节所述，是系统在面对特洛伊木马恶意程序时表现的脆弱。为了保持 Linux 内核的相对独立性，Android 在 Linux 之上的中间层添加了 MAC 式的权限控制——permission 机制，根据应用安装时用户的授权权限定义进行敏感数据和操作许可的判断。下面阐述的是 Android 的 permission 机制。

Android 对系统中的各种资源访问能力定义了详细的权限要求列表，例如，读取联系人的权限为 read_contacts，发送短信的权限为 send_sms，访问摄像头的权限为 camera 等。在用户下载安装应用的时候，应用列出所需的软件授权权限列表，用户必须同意给予授权，才能继续安装应用。应用安装成功后，针对限制应用的沙盒生成，该沙盒限制应用只能访问用户授权的能力访问范围。某种 Android 手机上，通过设置→应用程序→管理应用程序→应用程序信息→权限菜单进入查看应用的权限信息，看到的权限列表如图 2 所示。

2.4 沙盒技术困境

移动手机上沙盒技术一个困扰的问题是：应用的授权是否应该由用户负责，用户是移动手机和其数据的所有者，却不是专业的超级管理员。Android 的应用需要用户对应用进行授权，很大一部分用户可能不具备专业的技术背景理解每个权限代表的含义，因此无法做技术判断。例如，



图2 某种 Android 手机上的权限列表

当安装应用向用户请求读取联系人(read_contacts)以及访问网络的权限时,应用可能将联系人的信息发往网络,然而用户很可能不了解其中联系人隐私泄露的风险。

与此相比,在 iOS 的实现中,通过平台运营来弥补用户技术背景上的缺陷,在苹果公司的应用商店发布应用之前,必须经过苹果公司严格的审核,如静态代码分析、动态代码分析,保证应用没有侵害用户权益的恶意行为。另外,Android 的应用签名没有经过权威认证,当用户发现应用有恶意行为的时候,可能会找不到实施恶意行为的软件公司或者个人。苹果公司发现应用的恶意行为则可以直接将应用从苹果的应用商店下架。

3 应用运行时动态授权控制

应用运行时动态授权控制技术是指用户能够在应用安装后,修改应用对智能手机终端敏感能力的访问授权,在一些技术文档中,该功能被描述为权限吊销或者权限回收(permission revocation)。这种技术主要针对目前的智能手机终端中的软件授权不能在应用安装后收回敏感能力访问授权的问题。如 2.3 节所述,Android 用户在安装时必须全盘接受应用要求的权限列表,用户在安装后不能收回应用授权,若发现软件有恶意的行为只能卸载该软件。

然而在用户使用应用过程中,用户具有收回部分应用授权的强烈需求。例如,用户在使用某个微博应用的时候,发现该微博应用访问了照片信息,用户对此行为

并不认同,于是,用户需要收回对照片目录的访问权限,但不影响微博应用的基本使用,仅限制了需要访问照片目录的功能。

如上文所述,iOS 和 Android 的沙盒均是某种 MAC 技术,前者是操作系统层面实现的 MAC 技术,后者是操作系统之上的中间层实现的 MAC 技术。此处的讨论焦点在于,这两种 MAC 沙盒能否支持动态的权限定义修改?在 MAC 控制模式中,应用不能自行繁殖或者修改授权的策略定义,当系统发现当前应用需要某项授权来运作某种功能时,由系统服务弹出窗口与用户直接交互获取用户的授权,或者引导用户到配置窗口进行授权,亦即需要直接获取用户的真实操作意图。

3.1 iOS 的应用运行时动态授权控制

苹果公司的 iPhone 设备升级到 iOS 6,具有部分敏感能力的动态权限授权控制功能,包括系统的敏感数据或者能力,如 GPS 能力、通讯录访问、日历、提醒事项、照片、蓝牙共享,甚至是关键的社交应用,如 Twitter、Facebook、新浪、微博的账户等。在 iOS 6 的“设置—隐私”窗口中选择通讯录,可以看到请求访问通讯录信息的所有应用,用户具有可以选择关闭某个应用访问通讯录的能力。如图 3 所示,倘若用户关闭了微信的 QQ 通讯录的访问权限,仅影响微信中与通讯录相关的功能,并不影响微信其他功能的正常访问。此功能说明 iOS 6 的沙盒功能具有动态调整权限定义的能力。



图3 iOS 6 的动态权限授权控制

3.2 Android 的应用运行时动态授权控制

对 Android 操作系统来说,官方提供的版本并不提供动态的应用访问授权修改功能。一些基于 Android Open Source Project 开源社区提供的包含了整个操作系统的镜像的计算机数据文件,用于向手机的 ROM(read-only memory)芯片刷写安装系统的第三方 ROM 或者某些应

用,提供此功能,下面介绍几种实现方案。

第 1 种是需要修改系统的本身,对系统进行打补丁或者修改其权限机制。例如著名的 Android 非官方发布版本 CyanogenMod 在 7.0 的时候提供了应用授权回收的功能,在一台刷了 CyanogenMod 提供 ROM 的手机中,动态地管理应用程序的权限;SeAndroid 的开源项目,将权限吊销列入其 MAC 沙盒机制的实现范围,一些安全应用提供 Android 内核补丁加上一个用于管理的外部应用。这类权限回收方案的实现需要重新刷机,对普通用户而言无疑是个无法驾驭的方案。

第 2 种方案是需要对手机终端进行“root”。“root”是指破解移动设备,使之运行在超级用户权限下。从原理上说,权限回收功能仅能由系统提供,此功能的运作势必需要最高的超级用户权限,不可避免地需要将终端设备进行“root”。此方案破解了移动设备,为用户带来了其他的安全问题。

第 3 种方案其实是一种取巧的方案。取消权限的时候,需要用户主动卸载原来的应用,重新生成权限定义的 AndroidManifest.xml 文件,然后用户再安装新打包的应用。这种实现方案的问题是:有些预装的应用无法卸载,另外,修改了软件的官方发布的打包。

目前来看,Android 上应用访问权限回收的功能并不成熟。

4 智能手机的组件跨进程安全研究

在移动终端的实现中,存在着跨应用进程协作,例如,在查看用户最近邮件的功能中,可能需要启动联系人查看功能。由于邮件客户端功能和联系人功能分属两个不同的应用,此类功能存在着应用间的跨进程协作。

应用沙盒技术和应用间的通信是互相矛盾的,应用沙盒技术能够将移动应用对系统资源的访问限制在有限的资源集合内;而应用间通信则存在可能绕过应用沙盒,通过其他的应用间接获取对系统资源的使用。因此,应用间的通信需要被安全模型所监管。

4.1 iOS 的应用跨进程协作

苹果公司的 iOS 的应用沙盒依靠应用间通信的强力管制来实现,应用之间不直接进行通信或者共享数据,如果应用需要另一个应用的协作来完成一项任务,只能通过系统提供的 API 或者服务。

举一个例子来说,E-mail 应用和 TXT 编辑应用严格隔离,如果 E-mail 应用需要 TXT 文本编辑器打开和显示

一个文本的附件,TXT 文本编辑器需要声明自身能处理的文档类型,当用户点击 E-mail 的附件并选择了 TXT 文件编辑器的时候,E-mail 的 TXT 附件被 iOS 系统机制传送到 TXT 文本编辑应用的/Documents/Inbox 目录下,TXT 文本应用读取这个附件文件并展现给用户,当用户编辑这个文本附件时,TXT 文本编辑应用应将这个文件移动到本应用的数据目录下,因为沙盒规定/Documents/Inbox 只有读取和删除文件的权限,并没有写文件的权限。这是严格的沙盒结构的特例流程,苹果公司官方开发文档中需要专门文档进行阐述。

iOS 的设计为应用的跨进程协作设置了非常高的技术门槛,基本上杜绝了应用间互相勾结和利用其他应用提供的接口进行非法行为的可能性。

4.2 Android 的应用跨进程协作

4.2.1 Android 的应用组件及技术特点

4.2.1.1 Android 的应用组件构成

对 Android 操作系统来说,应用的跨进程协作无处不在。Android 操作系统上最小的可管理重用的协作单元是组件。应用、进程、组件之间的关系如图 4 所示。

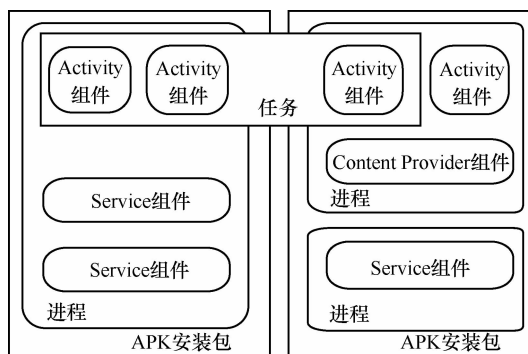


图 4 应用、进程、组件的关系

Activity 组件是包含显示窗口的基本的 Android 程序组件,是人机交互的基本单元。Activity 组件具有运行时的上下文和独特的生命周期,系统根据需要进行 Activity 组件调度。Activity 组件可以由用户点击系统桌面图标或者经由其他应用启动。

Service 组件不包含人机交互的界面,在后台执行某长期工作任务的组件,可以由应用内部的其他组件启动。Service 组件的另一用法是提供绑定 IPC (进程间通信)的接口,支持实现远程调用。Service 组件支持任意类型的后台任务,包括执行网络事务、播放音乐、下载后台文件等。

Content Provider 组件用于管理应用中可共享的数据,



如文件、数据库。

Broadcast Receiver 组件用于接收系统范围内的事件声明,例如,屏幕将被关黑、电池处在低电,或者其他程序的广播通知等。

Android 组件设计的关键特点是:程序可以启动另一个程序的组件。例如,某团队开发一个程序,需要实现用户打开摄像头并抓拍一张照片的流程,此功能已经有另一个应用的一系列组件完整实现,则不需要重新开发,只需要直接使用现有应用的组件即可。所谓的“直接使用”甚至不用将已有应用的组件链接进新的程序。

4.2.1.2 Android 组件通信机制

Android 有 3 种组件通信机制:Intent 机制、IPC Binding 机制和 Content Resolver。Intent 机制能够唤起 Android 的 3 种组件,Activity、Service、Broadcast Receiver 是 Android 组件间主要的通信方式,是一种间接的通信方式。IPC Binding 机制是使用远程函数调用的方法与 Service 组件通信,Content Resolver 用于与 Content Provider 组件通信。

Intent 是一种 Android 组件间的异步事件对象,携带了发送组件的“操作意图”。翻译成自然语言:发送 Intent 组件期望某个具体的组件(显式指定)或者期望系统帮忙找合适的组件(隐式指定)完成一项具体的任务,系统找到目标组件后,如果其没有处在运行状态,则负责创建组件并调度到运行状态执行指定的任务。Intent 机制保证了 Android 可以最大限度地重用系统或者其他安装应用的成熟组件。

Service 组件能够支持远程函数调用接口,这就是 IPC 的 Binding 用法,是 Service 组件与其他组件通信的一种特

殊通信方式。

Content Provider 组件不能由 Intent 唤醒,Content Provider 仅与 Content Resolver 通信,由其唤醒。Content Resolver 处理与 Content Provider 通信的整个过程,与 Content Provider 通信的组件不需要添加额外的逻辑,仅调用 Content Resolver 即可。

4.2.2 Android 的应用组件通信的安全研究

最近,业界注意到了 Android 应用安全的一个严重问题,应用间存在的勾结“Collusion”的恶意行为以及应用可能利用现有系统或者第三方应用的权限漏洞进行“Confused Deputy Attack”的攻击。

如同上文所述,Android 的应用组件间能够互相唤醒和调用以及存在着其他直接的沟通途径,组件间沟通通常并不在系统的 permission 机制控制范围内,因此具有严重的问题。

如图 5 所示,系统应用的某 Service 组件(此处称为 S)定义了访问者需具有访问智能手机终端的地理位置信息 access_fine_location 权限方能响应请求。应用 2 所在的应用沙盒具有 access_fine_location 的访问权限,因此其 Service 组件 S2 能够合法地唤起 S 组件。若应用 2 由于疏忽或者恶意的动机,没有对 S2 的访问者进行限制定义,实施恶意行为的应用 1 可以启动后台运行的 Service 组件 S1, S1 将通过应用 2 的 S2 组件间接发起对 S 组件的调用,获取用户的地理位置隐私信息。

此处的恶意攻击可以分属两种类别:第一种是被利用的组件是正常的组件,由于疏于保护而被恶意软件所用,

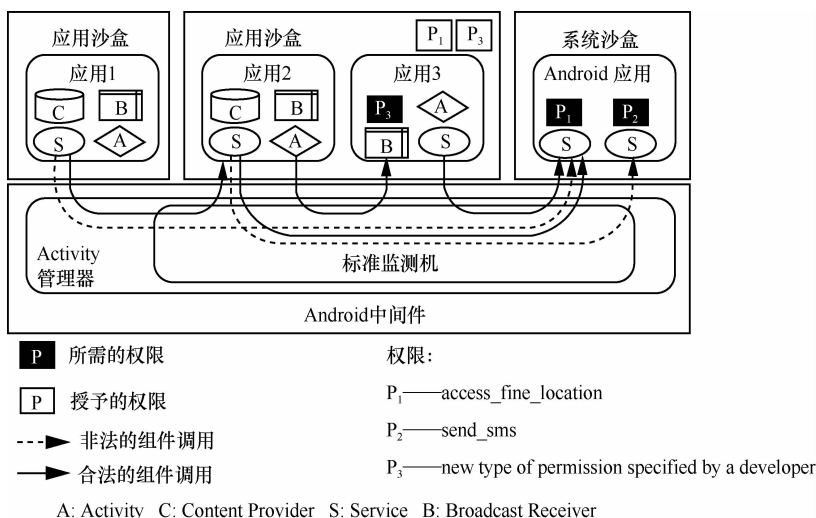


图 5 系统应用的某 Service 组件

这种攻击被称为 Confused Deputy Attack, 据研究, Android 欠缺完善的自我保护的组件有许多, 包括第三方的应用以及系统的应用, 如电话应用、音乐应用、配置应用等。第二种是被利用的组件与实施恶意行为的组件是有意配合和协作的关系, 称为 Collusion, 比如恶意攻击者首先发布一个功能正常的程序, 实现正常的功能引诱用户下载, 这个功能正常的程序获得了用户的授权, 能够读取敏感的数据, 此程序向特定的应用开放了调用权限, 然后再发布一个功能表面正常的程序, 引导用户给予访问互联网或者发送短信等的授权。后一个应用调用前一个应用, 则可以完成窃取用户的隐私数据并发送到互联网。应该说, 这两种攻击技术没有本质的不同。应用间可用的“协作”途径如下。

- 两个 Android 应用通过内部文件进行通信。第一个应用可以将需要通信的内容写到系统的日志中, 第二个应用在获取 read_logs 权限之后读取日志的内容。两个应用完成通信。
- 共享属性, Android 操作系统的共享属性是一种 <key,value> 的键值对, Android 提供了 API 进行读写。第一个应用获取了 mode_world_readable 和 mode_world_writable 模式允许其他应用读写其共享属性。
- 组件间的通信机制, 上文所述的 Intent 机制, 可以让一个应用间接地唤起另一个应用的 Activity、Service、Broadcast Receiver 的 3 种组件, 后两种组件可以运行在后台, 在用户没有察觉的情况下运行。IPC Binding 机制可以让一个应用调用另一个应用的后台运行的 Service 组件, 甚至一个应用可以共享 Content Provider 组件, 另一个应用则可以直接读写 Content Provider 组件中存储的信息。
- 直接使用本地网络套接字服务等。

“Confused Deputy Attack”和“Collusion”的安全问题涉及多个应用, 应用可能来源于同一个签名, 也可能来源于不同的签名, 应用组件只要有一个组件有问题, 就将成为系统的一个“后门”, 可以为其他的恶意应用所利用。由于涉及多个应用的协作, 单纯扫描、查询和分析某一个应用的行为, 可能无法发现此类问题的所在。

为了解决 Android 平台上利用组件间通信实施的恶意行为, 目前业界和学术界对 Android 平台提出了不少改进和扩展方案, 这些研究方案均在发展中, 并未合并进主流的 Android 版本。

例如, 其中一种典型的方案是, 若组件 A 具有权限访问组件 B, 组件 B 具有权限访问组件 C, 记录一条路径, 标记组件 A 对组件 C 具有访问权限。以此类推, 进行分析, 对 Android 平台的所有组件之间保存一张表, 描述每个组件的通达路径。将此表交付给用户审核, 去除用户不认同的访问路径之后, 由方案扩展的 Android 组件强制执行。此实现方法对用户的要求更高, 用户通常并无知识背景读懂复杂的组件权限访问路径表, 因此, 此方案不能算是成功的方案。

5 敏感信息的流向跟踪技术

5.1 实现的基本原理

移动互联网应用中, 用户的隐私信息, 如联系人、地理位置、电话号码、通话记录等信息都很容易被应用窃取, 并上传到某个网络服务器上。目前一般的技术手段无法解决这个问题。例如, 对于 Android 的平台, 如果应用要求有访问联系人的权限以及访问互联网的权限, 应用就有可能将联系人的信息上传到互联网。

为了更彻底地解决这个问题, 学术界出现了利用移动终端上的敏感信息进行流向跟踪的技术。目前, 主要有 TaintDroid 开源项目、SeAndroid 的开源项目等。下面以 TaintDroid 项目为例, 介绍移动手机上的敏感信息流向跟踪技术。

TaintDroid 的基本思路是: 通过改造 Android 上的 Dalvik 虚拟机(Android 上的 Java 虚拟机实现), 使之对包含了敏感信息的变量进行标记。例如, 如果变量数据的原始来源是用户的联系人、邮件、日程等敏感信息, 变量是被敏感信息“污染”的, 在被“污染”的变量上设置敏感信息标记。当变量被复制时, 敏感标记同样进行了繁殖, 如果包含了敏感信息的变量被作为参数发送出系统外, 则向用户告警, 如果用户不允许此操作, 则进行拦截。

TaintDroid 修改 Dalvik 虚拟机的实现, 对基础的函数局部变量、函数参数、类静态变量、类实例变量、数组变量等, 添加额外的标记数据位, 标记存储变量是否为敏感信息; 对函数局部变量、函数参数需修改原本函数的压栈结构, 为每个变量添加额外的标记位; 对类的静态变量、类实例变量需要修改类的实现添加额外的标记位。数组变量则是整个数组共享一个敏感信息标记位, 即当数组变量有部分数据为敏感数据时, 整个数组变量为敏感的数据。

对 Dalvik 虚拟机运作的 DEX 机制指令集的每个指



Op Format	Op Semantics	Taint Propagation	Description
<i>const-op</i> $v_A C$	$v_A \leftarrow C$	$T(v_A) \leftarrow \phi$	clear v_A taint
<i>move-op</i> $v_A v_B$	$v_A \leftarrow v_B$	$T(v_A) \leftarrow T(v_B)$	set v_A taint to v_B taint
<i>move-op</i> $R v_A$	$v_A \leftarrow R$	$T(v_A) \leftarrow T(R)$	set v_A taint to v_B return taint
<i>return-op</i> v_A	$R \leftarrow v_A$	$T(R) \leftarrow T(v_A)$	set return taint (ϕ if void)
<i>move-op</i> $E v_A$	$v_A \leftarrow E$	$T(v_A) \leftarrow T(E)$	set v_A taint to exception taint
<i>throw-op</i> v_A	$E \leftarrow v_A$	$T(E) \leftarrow T(v_A)$	set v_A exception taint
<i>unary-op</i> $v_A v_B$	$v_A \leftarrow v_B \otimes$	$T(v_A) \leftarrow T(v_B)$	set v_A taint to v_B taint
<i>binary-op</i> $v_A v_B v_C$	$v_A \leftarrow v_B \otimes v_C$	$T(v_A) \leftarrow T(v_B) \cup T(v_C)$	set v_A taint to v_B taint $\cup$ v_C taint
<i>binary-op</i> $v_A v_B$	$v_A \leftarrow v_A \otimes v_B$	$T(v_A) \leftarrow T(v_B) \cup T(v_B)$	update v_A taint with v_B taint
<i>binary-op</i> $v_A v_B C$	$v_A \leftarrow v_B \otimes C$	$T(v_A) \leftarrow T(v_B)$	set v_A taint to v_B taint
<i>binary-op</i> $v_A v_B C$	$v_B[v_C] v_B \leftarrow v_A$	$T(v_B[v_C]) \leftarrow T(v_B[v_C]) \cup T(v_A)$	update array v_B taint with v_A taint
<i>aput-op</i> $v_A v_B v_C$	$v_A \leftarrow v_B[v_C]$	$T(v_A) \leftarrow T(v_B[v_C]) \cup T(v_C)$	set v_A taint to array and index taint
<i>aput-op</i> $v_A v_B v_C$	$f_B \leftarrow v_A$	$T(f_B) \leftarrow T(v_A)$	set field f_B taint to v_A taint
<i>sput-op</i> $v_A f_B$	$v_A \leftarrow f_B$	$T(v_A) \leftarrow T(f_B)$	set v_A taint to field f_B taint
<i>sput-op</i> $v_A f_B$	$v_B(f_C) \leftarrow v_A$	$T(v_B(f_C)) \leftarrow T(v_A)$	set field f_C taint to v_A taint
<i>iput-op</i> $v_A v_B f_C$	$v_A \leftarrow v_B(f_C)$	$T(v_A) \leftarrow T(v_B(f_C)) \cup T(v_B)$	set field f_B taint to v_A taint
<i>iput-op</i> $v_A v_B f_C$			set v_A taint to field f_C and object reference taint

图6 TaintDroid的敏感标记位繁殖逻辑

令,TaintDroid添加敏感标记位的繁殖逻辑,当发现敏感信息的变量、对象、常量等被复制和运算时,按照繁殖的规则,敏感信息标记位繁殖复制到运算涉及的其他变量。

例如, $a=b$ 语句,机器运算指令为 *move-op* $a b$,变量 b 如果为包含敏感信息的变量,本语句运算完成后, a 变量也将被设置为敏感信息的标记位。其实际意义在于,如果变量 b 包含敏感信息,如用户的银行账号,即 b 是被敏感信息“污染”的变量,因此,在程序的运算过程中,如果 b 被赋值给另外的变量 a ,那么 a 也是被敏感信息“污染”的变量。

图6是TaintDroid的敏感标记位繁殖逻辑示意^[4],其中 v_x 为寄存器变量,包括函数局部变量、参数变量; f_x 为类的静态变量,类的实例变量表示为 $v_x f_x$,其中的 v_x 为实例对象引用; R 为返回变量, E 为异常变量(exception), C 为常量; $v_x[\cdot]$ 为数组的变量。

5.2 实现的技术瓶颈

敏感信息流向的跟踪技术是一种复杂而尚待检验的技术,在这种技术实现中,有一些尚待检验和有待成熟的技术瓶颈问题。

(1)敏感信息标记繁殖规则的准确性问题

变量敏感信息标记如果在程序运算的过程中不能准确地繁殖,用户的隐私就很可能被泄漏。

(2)远程调用过程中的敏感信息标记位的传递

Dalvik虚拟机的远程调用使用的是Binder IPC技术,需要修改Android的Binder服务类库来携带进程远程调用中的敏感标记信息。

(3)在本地调用(native call和JNI)的敏感信息中传递

由于无法跟踪Dalvik虚拟机之外的本地方法调用过程中的敏感标记繁殖,为了保证敏感信息不外泄,有两种方法:第一种方法认为所有访问本地方法调用的变量均是被“污染”的,需要添加标记,这无疑粗放地扩大了“污染”的范围;第二种方法对本地系统方法调用的函数进行辨别,这种辨别需要对系统调用的每个函数进行分析,这种工作难度比较大,并需要在系统每个版本更新的时候,更新系统函数调用的分析库。

(4)在文件系统以及外部存储过程的敏感信息传递

敏感信息的变量可能被存储到文件系统(Android上的文件系统是YAFFS2)和设备的SD卡存储系统(如ext 2)时丢失敏感标记,为了避免这种情况,需要扩展文件系统的属性,保持敏感信息标记,这是待深化研究的领域。

6 结束语

随着移动互联网的迅猛发展,从各大安全公司以及研究机构最近发布的智能手机安全报告来看,智能手机的安全问题愈演愈烈,正在成为制约移动互联网发展的关键瓶颈之一。从本文的研究发现,应用安全的关键技术均严重依赖于移动操作系统提供的平台基础,如沙盒隔离机制、应用组件构成等。因此,应用安全的提升需要推动移动操作系统往更安全的方向演进。在未来的进一步研究中,可以操作系统的底层安全机制为深化研究的方向,推动从操作系统到上层应用的移动终端一体化安全。

参考文献

1 DAC 的维基百科. http://en.wikipedia.org/wiki/Discretionary_access_control

2 MAC 的维基百科. http://en.wikipedia.org/wiki/Mandatory_access_control

3 Android 开发官方文档. <http://developer.android.com/guide/components/index.html>

4 Apple iOS 开发官方文档. <https://developer.apple.com/library/iOS/navigation/index.html>

5 Robert N M Watson. New approaches to operating system security extensibility. 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, 2012

6 William E, Peter G, Byung-Gon C, *et al.* TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. 9th USENIX Symposium on Operating Systems Design and Implementation, Vancouver, BC, Canada, 2010

7 Dino A, Dai Z. Apple iOS 4 Security Evaluation, 2011

8 Sven B, Lucas D, Alexandra D, *et al.* XManDroid: a New Android Evolution to Mitigate Privilege Escalation Attacks. System Security Lab Technische University at Darmstadt, Germany, 2011

9 Rafael F, Christian B, Christoph K, *et al.* Android OS Security: Risks and Limitations. AISEC Technical Reports, AISEC -TR - 2012-001, 2012

[作者简介]



罗喧,女,学士,中国电信股份有限公司广东研究院工程师,主要从事移动互联网开发工作。

梁柏青,男,学士,中国电信股份有限公司广东研究院部门技术总监、高级工程师,“核心电子器件、高端通用芯片及基础软件产品”国家科技重大专项基金资助项目“移动智能终端操作系统研发”的负责人,主要从事电信网络技术研究,交换设备、综合数据网络产品和电信终端的设计和研发工作。

潘军彪,男,中国电信股份有限公司广东研究院政企客户产品开发部工程师,主要研究方向为互联网平台与终端技术。

黄粤,男,学士,中国电信股份有限公司广东研究院政企客户产品开发部工程师,主要从事政企产品和行业应用解决方案的研究与开发工作。

(收稿日期:2013-04-15)

·简讯·

2013 年 3 月通信业主要指标完成情况(二)

指标名称	单位	本月末到达	比上年末净增	本月净增
固定电话用户合计	万户	27 640.6	-174.7	-91.7
公用电话用户	万部	2 316.9	-30.3	-11.2
城市电话用户	万户	18 873.6	-19.8	-44.5
住宅电话用户	万户	10 913.4	-99.7	-48.6
农村电话用户	万户	8 767.0	-154.9	-47.3
住宅电话用户	万户	7 153.1	-155.7	-56.8
移动电话用户合计	万户	114 572.1	3 356.6	1 367.8
其中:3G 用户	万户	27 726.7	4 446.3	1 735.1
互联网拨号用户	万户	569.3	-0.5	-2.2
互联网宽带接入用户	万户	18 146.9	628.6	256.2
其中:xDSL 用户	万户	11 472.0	-429.1	-37.6
移动互联网用户	万户	81 739.2	5 302.8	1 394.1
固定电话普及率	部/百人	20.5		
移动电话普及率	部/百人	84.9		

(来源:工业和信息化部运行监测协调局)