软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(1):45−71 [doi: 10.13328/j.cnki.jos.004914]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

Android 安全研究进展*

卿斯汉[1,2,3]

[1](中国科学院 软件研究所,北京 100190)

[2](信息安全国家重点实验室(中国科学院 信息工程研究所),北京 100093)

[3](北京大学 软件与微电子学院,北京 102600)

通讯作者: 卿斯汉, E-mail: qsihan@ss.pku.edu.cn

摘 要: Android 是目前最流行的智能手机软件平台,报告称,2014 年,Android 的销售量占到全球份额 81%的绝对优势,首次达到 10 亿部.其余如苹果、微软、黑莓与火狐等则远远落在后面.与此同时,Android 智能手机的日益流行也吸引了黑客,导致 Android 恶意软件应用的大量增加.从 Android 体系结构、设计原则、安全机制、主要威胁、恶意软件分类与检测、静态分析与动态分析、机器学习方法、安全扩展方案等多维角度,对 Android 安全的最新研究进展进行了总结与分析.

关键词: Android;安全机制;恶意软件;静态分析与动态分析;安全扩展方案

中图法分类号: TP309

中文引用格式: 卿斯汉. Android 安全研究进展.软件学报,2016,27(1):45−71. http://www.jos.org.cn/1000-9825/4914.htm

英文引用格式: Qing SH. Research progress on Android security. Ruan Jian Xue Bao/Journal of Software, 2016,27(1):45−71 (in Chinese). http://www.jos.org.cn/1000-9825/4914.htm

Research Progress on Android Security

QING Si-Han[1,2,3]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Information Security (Institute of Information Engineering, The Chinese Academy of Sciences), Beijing 100093, China)

[3](School of Software and Microelectronics, Peking University, Beijing 102600, China)

Abstract: Android is a modern and most popular software platform for smartphones. According to report, Android accounted for a huge 81% of all smartphones in 2014 and shipped over 1 billion units worldwide for the first time ever. Apple, Microsoft, Blackberry and Firefox trailed a long way behind. At the same time, increased popularity of the Android smartphones has attracted hackers, leading to massive increase of Android malware applications. This paper summarizes and analyzes the latest advances in Android security from multidimensional perspectives, covering Android architecture, design principles, security mechanisms, major security threats, classification and detection of malware, static and dynamic analyses, machine learning approaches, and security extension proposals.

Key words: Android; security mechanism; malware; static and dynamic analyses; security extension proposal

在智能移动终端如火如荼发展的同时,其安全态势也日益严峻.Alcatel Lucent 旗下的 Motive Security Labs 发布的恶意软件分析报告[1]指出,2014 年,全球受恶意软件感染的移动设备为 1 600 万台,占全部移动设备的 0.68%,同比增长了 25%,高于 2013 年的 20%.图 1 说明,自 2012 年 12 月~2014 年 12 月之间移动设备感染率增

* 基金项目: 国家自然科学基金(61170282)
Foundation item: National Natural Science Foundation of China (61170282)
收稿时间: 2015-06-20; 修改时间: 2015-08-31; 采用时间: 2015-09-11; jos 在线出版时间: 2015-10-16
CNKI 网络优先出版: 2015-10-16 11:53:29, http://www.cnki.net/kcms/detail/11.2560.TP.20151016.1153.001.html

长的情况.

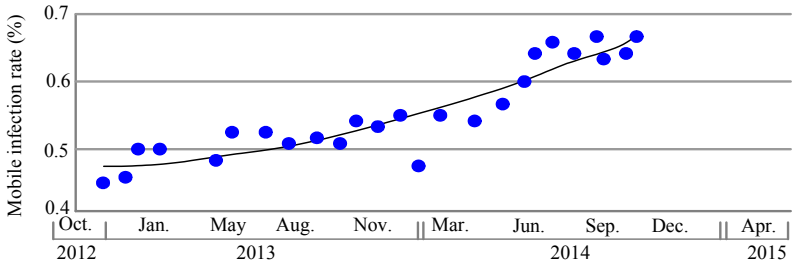

Fig.1 Mobile infection rate since December 2012

图 1 2012 年 12 月以来移动设备感染率

Android 是目前应用最广的智能手机操作系统,它的设计兼顾系统的性能、可用性、安全性与开发方便性,受到广大用户的欢迎.据权威分析机构 Strategy Analytics 的统计[2],2014 年,全球智能手机的年销售增长率大于 30%,达到超记录的 13 亿部,其中,Android 的销售量占到全球份额 81%的绝对优势,首次达到 10 亿部.其余如苹果、微软、黑莓与火狐等则远远落在后面.另一方面,Android 系统的开放性也受到应用开发者的青睐,但同时也带来更多的安全问题.Android 设备和 Windows PC 已经成为恶意攻击的主要对象,遭恶意软件攻击的次数已经趋同.2014 年,Android 恶意软件样本的增长率为 161%.图 2 是自 2012 年 6 月以来的统计[1].

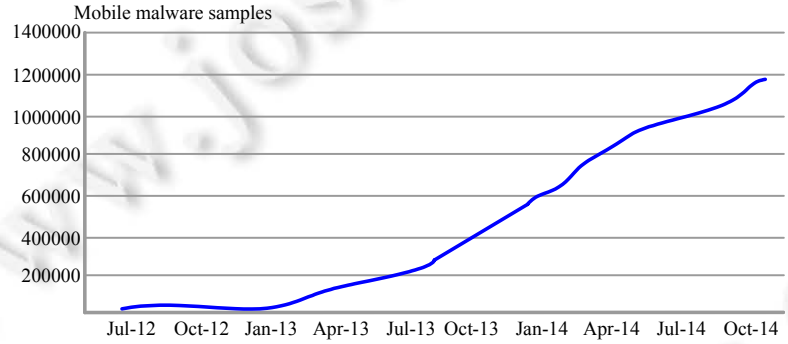

Fig.2 Android malware samples continue growth

图 2 Android 恶意软件样本量持续增加

然而,Android 应用的迅猛发展和安全问题的日益突出与反制措施极不协调.Zhou 等人[3]的研究表明,4 个有代表性的移动反病毒软件:AVG(AVG antivirus free v2.9),Lookout(lookout security & antivirus v6.9),Norton (norton mobile security lite v2.5.0.379)和 TrendMicro(TrendMicro mobile security personal edition v2.0.0.1294)在最佳和最坏情形下只能分别检测出 79.6%和 20.2%的恶意软件.

在这种形势下,近年来,关于 Android 安全的研究大幅度增加.来自会议、期刊和网络的论文覆盖面很广:智能手机(Android,iOS,Windows Phone 等)安全综述如文献[4−6]、Android 安全综述如文献[7−13]、某类问题的综述——如针对 Android 恶意软件[14−31];Android 的权限机制[32−50];机器学习在 Android 中的应用[51−61]等.更多的研究是关于某个具体方向的分析和讨论以及对某个具体问题的深入分析.例如:Felt 等人[33]对 Google 官方文件没有仔细阐述的 Android 权限实现机制进行了剖析,给出了权限与 API 之间的映射关系.此外,还有厂商提供的应用工具与安全工具等,如 Google 提供的反编译工具 Apktool[62]和 FireEye 公司提供的基于“多向量虚拟执行(multi-vector virtual execution,简称 MVX)”引擎的移动威胁保护工具(FireEye mobile threat prevention,简称 MTP)[63].

Android 的安全问题并非都是其独有的,Android 继承了许多发生过的类似安全问题,自然地也继承了其底

层 Linux 中的安全问题.例如,“*Setuid*()-RLIMIT_NPROC 交互”型漏洞经常发生在 Linux(以及 Unix)中各种 root 守护进程与 setuid-root 程序之中,该漏洞源于没有正确检查 *setuid*()函数的返回值.*setuid*()执行成功或失败时,分别返回 0 和−1.当程序以 root 身份运行并正常执行 *setuid*(*UID*)后,将降低权限为普通用户.但因未检查 *setuid*()的返回值,当 *setuid* 失败时,程序仍将继续以 root 身份运行,就会产生许多安全问题.Zimperlich[64]恶意软件(参看第 3.1.2 节)正是利用了上述漏洞,成功地攻击了 Android 系统.

下面,本文将从 Android 体系结构、设计原则、安全机制、主要威胁、恶意软件分类与检测、静态分析与动态分析、机器学习方法、安全扩展方案等多维角度,对 Android 安全的研究现状与进展进行全面的总结与分析.最后,对全文进行总结.

1 Android 体系结构

1.1 Android软件栈

Android 的软件栈具有清晰的 3 层结构[65],由底向上是 Linux 内核、Android 中间件和应用层,如图 3 所示.

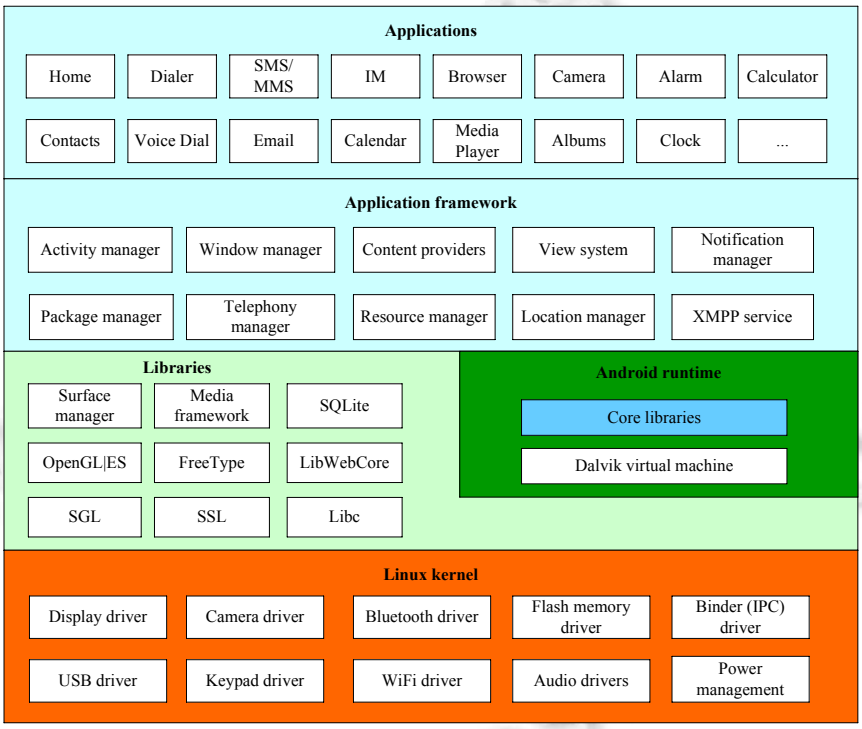

Fig.3 Android software stack

图 3 Android 软件栈

Linux 内核提供最基本的功能,包括内存管理、进程调度、设备驱动器和文件系统.所有的设备资源,例如照相与摄像功能、GPS 数据、蓝牙功能、通话功能、网络连接等,都通过操作系统进行访问.

Android 在内核层实现了基于 Binder(OpenBinder[66]的精简版本)的轻量级进程间通信(inter-process communication,简称 IPC),这是应用组件之间进行通信的主要 IPC 机制,在 Android 的术语中称为组件间通信(inter-component communication,简称 ICC).Android 为 ICC 设计了一种特定的接口定义语言(Android interface definition language,简称 AIDL),定义用于远程调用的 method 和 field.例如,可以通过 ICC 调用绑定远程服务.

Linux 之上的中间件层由本地类库、Android 运行环境和 Android 应用架构组成.本地类库提供诸如图形处

理等基本功能.Android 运行环境由核心 Java 类库与 Dalvik 虚拟机组成.Android 应用架构是用 C/C++或 Java 编写的系统应用,由 System Content Provider 和 System Service 构成.其中,System Content Provider 是基本数据库;System Service 包括联系人 app、前贴板、系统设置、Notification Manager、Resource Manager、Location Manager 等,System Service 提供控制设备硬件和获取平台状态信息(如位置、网络状态)等功能.

最上面的应用层由两部分组成:一是内置的基本应用,包括桌面(home)、拨打电话(phone)、E-mail、日历(calendar)、Web 浏览器(browser)和联系人(contacts)等;二是用户安装的用 Java 编写的第三方应用,但由于性能的原因,许多应用也包括本地代码(C/C++),可以通过 Java 本地接口(Java native interface,简称 JNI)调用.

1.2 Android的组件

Android 有 4 种类型的组件(component).

(1) Activity(用户界面)是应用程序与用户进行人机交互的可视化用户界面,包含很多与用户交互的构件,如按钮、文本输入框等.一个 Android 应用程序可以有多个 Activity;

(2) Service(后台进程)是 Android 系统运行在后台的服务,不提供用户界面;

(3) Content Provider(内容提供器)是一种 SQL-数据库,用于为应用程序提供数据,同时为应用程序中的数据共享提供支持;

(4) Broadcast Receivers(广播接收器)是接收广播消息的邮箱.

1.3 Intent与组件通信

Android 系统中不同组件之间的通信主要是通过 Intent 实现的,Intent 是一个包含目标组件地址和数据的消息对象.Android API 定义了接收 Intent 和通过其中的信息启动 Activity(startActivity(Intent))、启动 Service (startService(Intent))和广播消息(sendBroadcast(Intent))的方法(method).调用这些方法相当于通知 Android 系统开始调用目标应用中的执行代码,在 Android 术语中,这种组件间通信的过程称为“动作(action)”.亦即,一个 Intent 对象定义执行一个“动作”的意图(intent),这就是“Intent”名称的缘由.

Android 系统发送 Intent 对象的方式有两种类型:(1) 显式类型,发送时在 Intent 对象的字段中标注目标组件名称,即,该 Intent 对象会到达该指定组件;(2) 隐式类型,发送时只设置 Intent 对象中的“动作(action)”字段,发出后可能有多个程序通过 intent-filter 机制能够处理该动作.应用程序在组件中可以声明 intent-filter,定义该组件可以接收的动作和处理的数据类型.这种使用“隐式类型”的 Intent 寻址机制的灵活性是 Android 系统最为强大的功能之一,但同时也会带来安全隐患.

Android 组件的通信如图 4 所示.

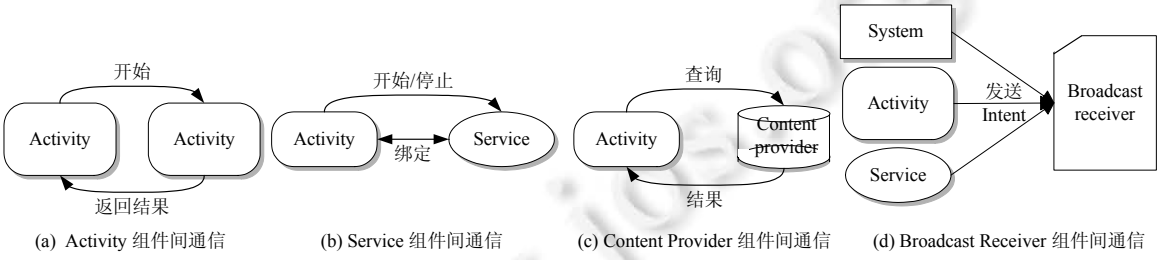

Fig.4 Android inter-component communication

图 4 Android 组件间通信示意图

1.4 Android系统架构

Android 的系统架构[67]如图 5 所示,除前面阐述的软件栈之外,还有硬件的支持.Android 支持两种指令集架构:ARM 和 x86.

硬件抽象层(hardware abstraction layer,简称 HAL)是 Android 系统调用设备驱动层的标准接口,避免涉及驱

动器和硬件的底层实现.HAL 通常以.so 文件形式实现,.so 文件是 Linux 操作系统的动态链接库文件格式,相当于 Windows 操作系统的.dll 动态链接库文件格式.

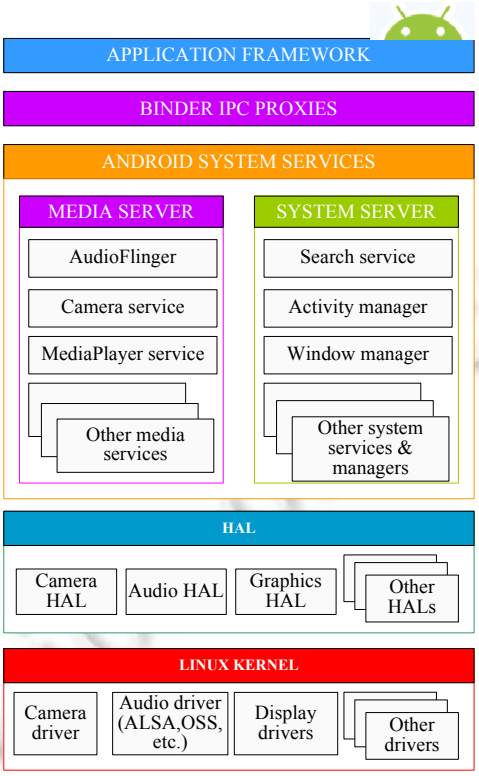

Fig.5 Android system architecture

图 5 Android 系统架构

1.5 Android的设计原则

笔者认为,Android 的设计遵循了下面 3 个原则.

(1) 开放性:在与苹果手机等的激烈竞争中,为吸引用户、第三方开发商与应用程序开发者所做出的选择,从目前的市场销售量与发展趋势看,无疑取得了很好的效果;

(2) 资源有限性:由于手机应用的资源(内存、电池等)有限,必须进行相应的重新设计.此外,由于某些系统受到版权保护,也需要进行重新设计,例如 Bionic(非 POSIX libc);

(3) 简单性:为了方便用户,Android 系统设计采取了“简单、快速与易用”的原则,在安全性方面作了折中.

值得注意的是:尽管 Android 的基础是 Linux,但对 Linux 作了大幅度修改,因此,Android 不是 Linux.概括地说,Android 对 Linux 的内核级增强包括 alarm 驱动器、ashmem(Android 共享内存驱动器)、binder 驱动器(Android 特有的轻量级进程间通信机制)、电源管理、低内存管理器(low memory killer,简称 LMK)和内核调试器(kernel debugger)等.其中,alarm 驱动器通过定时器唤醒休眠设备;ashmem 使 Android 应用可以高效地在内核级共享有限的内存;电源管理构建在标准 Linux 电源管理(PM)之上,采用更加有效的省电策略.Android 对 Linux 的用户级增强包括 Bionic(非 POSIX C 程序库)、预链接的系统库、Dalvik 虚拟机和本地类库等.此外,Android 使用在有限资源条件下优化的 YAFFS(yet another flash file system) flash 文件系统.

1.6 Dalvik虚拟机

Dalvik 虚拟机是 Android 系统的核心组成部分之一,它与 Java 虚拟机(JVM)不同,是专门为资源有限的移动

终端设计的,可使智能手机同时有效地运行多个虚拟机.每个应用程序都作为一个 Dalvik 虚拟机实例,在自己的进程中运行.Dalvik 虚拟机的可执行文件格式是.dex(Dalvik executable),是专门为 Dalvik 设计的一种压缩格式.与基于栈的 JVM 不同,具有更高效率的 Dalvik 虚拟机是基于寄存器的.Java 源代码编译成.class 文件后,通过 Android SDK 中的“dx”工具转换为 Dalvik 虚拟机可执行的.dex 文件.每个.class 文件中只包含一个 class,但一个 dex 文件可以包含多个 class.

Dalvik 虚拟机中有一个重要的虚拟机进程 Zygote,称为虚拟机实例的孵化器.每当系统要求执行一个 Android 应用程序时,Zygote 就会 fork 出一个子进程来执行该应用程序.Zygote 是系统启动时产生的,它会完成虚拟机的初始化、库的加载等操作.如果系统需要创建一个新的虚拟机实例,Zygote 就通过 fork 自身,以最快的速度提供给系统.对于一些只读的系统库,所有虚拟机实例都和 Zygote 共享一块内存区域,极大地节省了内存开销.Dalvik 虚拟机有一套自身专用的指令集,并规定了这套指令集的指令格式与调用规范,称为 smali 语言.

2 Android 的安全机制

2.1 Android安全模型

Android 的安全模型由 3 个部分组成:Linux 安全机制、Android 本地库及运行环境安全与 Android 特有的安全机制,见表 1.

Table 1 Android security model

表 1 Android 安全模型

安全模型的组成部分	安全机制
Linux 内核	POSIX 用户 文件访问控制
Android 本地库及运行环境	内存管理单元 强类型安全语言 移动设备安全
Android 特定安全机制	权限机制 签名机制 组件封装 Dalvik 虚拟机

2.1.1 Linux 安全机制

Android 继承了 Linux 的安全机制,主要有:

(1) POSIX(portable operating system interface of unix)用户.

每个包(.apk)文件安装时,Android 会赋予该文件唯一的 Linux(POSIX)用户 ID.因此,不同的包代码不可能运行于同一进程,相当于系统为每个应用建立了一个沙盒,不论调用应用程序的方式如何,一个应用程序始终运行在属于自己的进程中,拥有固定的权限.

为使两个应用程序共享权限,它们必须共享相同的 ID,并通过 sharedUserID 功能实现.

(2) 文件访问控制.

Android 系统文件与应用文件的访问控制继承了 Linux 的权限机制.每个文件都绑定 UID(用户 ID)、GID(用户组 ID)和 rwx 权限,用于进行自主访问控制(discretionary access control,简称 DAC).一般地,Android 系统文件的拥有者是“系统”用户或“根”用户.为了增强安全性,所有的用户和程序数据都存储在数据分区,与系统分区隔离.当 Android 系统处于“安全模式”时,数据分区的数据不会加载,便于系统进行有效的恢复管理.此外,系统镜像设置为只读.

2.1.2 Android 本地库及运行环境安全

Android 本地库及运行环境也提供了安全保障,主要有:

(1) 内存管理单元(memory management unit,简称 MMU).

该硬件设备为进程分配不同的地址空间(虚拟内存),是一种隔离进程的有效方法.进程只能访问自己的内存页,而不能访问其他进程的内存空间.

(2) 强制类型安全.

类型安全(type safety)是编程语言的一种特性,它强制变量在赋值时必须符合其声明的类型,防止变量被错误或不恰当地使用.类型转化错误或缺少类型安全边界检查,是产生缓冲区溢出攻击的主要原因.Android 使用强类型 Java 语言,通过编译时的类型检查、自动的存储管理和数组边界检查保证类型安全.此外,Binder 也是类型安全的,经 Binder 传送的数据元类型由 AIDL 定义,保证类型在跨越进程边界时保持不变.

(3) 移动设备安全.

电话系统的基本属性集来自识别用户、监督使用和收费的需求.一个更为通用的术语是所谓的“AAA”,即,认证(authentication)、授权(authorization)和记账(accounting).Android 借鉴智能手机设计的这些典型安全特征,认证和授权通过 SIM 卡及其协议完成,SIM 卡中保存使用者的密钥.

2.1.3 Android 特定安全机制

Android 独有的安全机制包括:

(1) 权限机制.

这是 Android 中最重要的安全措施之一,将在第 2.2 节中详细阐述.

(2) 组件封装.

通过组件(activity,service,content provider 和 broadcast receiver)封装,可以保证应用程序的安全运行.若组件中的功能 exported 设置为 false,则组件只能被应用程序本身或拥有同一 UID 的应用程序访问,此时,该组件称为私有组件;若 exported 设置为 true,则组件可被其他应用程序调用或访问,此时,该组件称为公开组件.

(3) 签名机制.

为便于安装,Android 将应用程序打包成.apk 文件.一个.apk 文件不仅包含应用的所有代码(.dex 文件),也包含应用所有的非代码资源,如图片、声音等.Android 要求所有的应用程序(代码和非代码资源)都进行数字签名,从而使应用程序的作者对该应用负责.只要证书有效,且公钥可以正确地验证签名,则签名后的.apk 文件就是有效的.

(4) Dalvik 虚拟机.

如第 1.6 节所述,每个应用程序都作为一个 Dalvik 虚拟机实例在自己的进程中运行.因此,Dalvik 虚拟机与 POSIX 用户安全机制一起构成了 Android 沙盒机制.

2.2 Android权限机制

权限管理是 Android 系统中最重要的安全措施之一,经过了精心设计与考量.为达到方便、易用和简单、快速的目标,Android 在安全性方面做出了一定程度的牺牲,即,决定采取粗粒度的权限管理机制[68].

Android 的权限管理遵循“最小特权原则[69],即,所有的 Android 应用程序都被赋予了最小权限.一个 Android 应用程序如果没有声明任何权限,就没有任何特权.因此,应用程序如果想访问其他文件、数据和资源就必须在 AndroidManifest.xml 文件中进行声明,以所声明的权限去访问这些资源.否则,如果缺少必要的权限,由于沙箱的保护,这些应用程序将不能够正常提供所期望的功能与服务.

所有应用程序对权限的申请和声明都被强制标识于 AndroidManifest.xml 文件之中,通过〈permission〉,〈permission-group〉,〈permission-tree〉等标签指定.需要申请某个权限,可以通过〈uses-permission〉指定.应用程序申请的权限在安装时提示给用户,用户可以根据自身需求和隐私保护的考量决定是否允许对该应用程序授权.

由于基于 Linux 内核,Android 系统中的权限分为以下 3 类.

(1) Android 手机所有者权限;

(2) Android ROOT 权限.类似于 Linux,这是 Android 系统中的最高权限.如果拥有该权限,就可以对 Android 系统中的任何文件、数据、资源进行任意操作.所谓“越狱(JailBreak)”,就是令用户获得最高的 ROOT 权限;

(3) Android 应用程序权限.该权限在 AndroidManifest 文件中由程序开发者声明,在程序安装时由用户授权.共有下述 4 类不同的权限保护级别(protection level).

(i) normal.风险较低的权限,只要申请即可使用,无需用户授权,例如“VIBRATE”权限(使设备震动);

(ii) dangerous.风险较高的权限,安装时需要用户确认才能使用,例如“PHONE CALLS”(打电话),“INTERNET”(访问 Intent)和“SEND SMS”(发短信)权限;

(iii) signature.仅当申请该权限的应用程序与声明该权限的程序使用相同的签名时,才赋予该权限;

(iv) signatureOrSystem.仅当申请该权限的应用程序位于相同的 Android 系统镜像中,或申请该权限的应用程序与声明该权限的程序使用相同的签名时,才赋予该权限.

Android 权限机制是通过其中间件层实现的,由其访问监控器(reference monitor)对 ICC 调用实施强制访问控制(mandatory access control,简称 MAC).运行时,当组件请求 ICC 调用时,访问监控器核查该组件是否具有相应的权限.在图 6[11]所示的例子中,应用 1 中的组件 *A* 试图访问应用 2 中的组件 *B* 和 *C*,但比较 *B* 和 *C* 的访问权限标签和赋予应用 1 的访问权限标签集后,访问监控器允许 *A* 访问 *B*,但拒绝 *A* 访问 *C*.

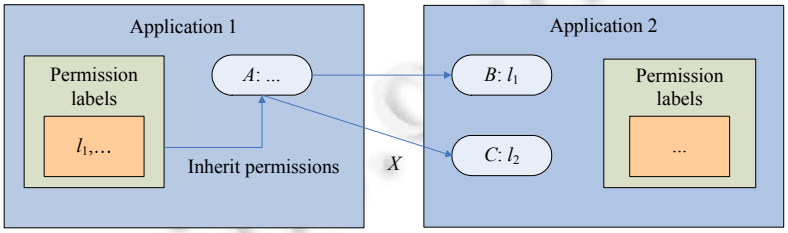

Fig.6 An example of Android permission mechanism enforcement

图 6 Android 权限机制实现举例

图 7[70]描述了 Android 的通信信道与相应的访问控制机制:在中间件层,Android 通过访问监控器对 ICC 调用进行强制访问控制;在内核层,Android 继承了 Linux 的安全机制,对文件系统和默认 Linux 进程间通信实施自主访问控制.同时,对网络套接字(socket)实施强制访问控制.

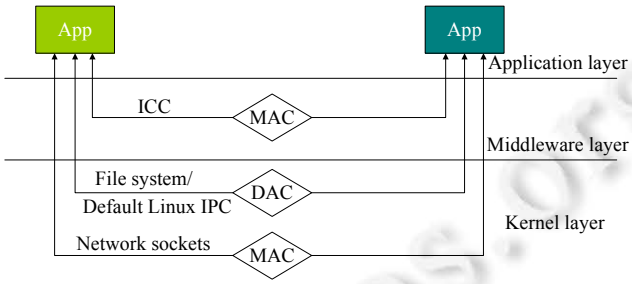

Fig.7 Android communication channel and its access control mechanism

图 7 Android 通信信道与相应的访问控制机制

2.3 Android权限机制的安全缺陷

迄今,研究者对 Android 的权限机制提出了诸多批评,对其安全缺陷进行了深入分析.概括地说,Android 权限机制的安全缺陷主要表现在以下几个方面.

(1) 粗粒度的授权机制

如前所述,Android 的权限机制是粗粒度的,即:要么赋予权限,要么拒绝授权.当某个应用申请了不适当或过多权限时,多数情形下用户都会选择授权,而很多安全问题便由此而来.因此,研究者提出了各种较细粒度 Android 授权机制的改进方案[71−73].

(2) 粗粒度权限

值得指出的是:不仅 Android 的权限管理机制是粗粒度的,大多数 Android 权限也是粗粒度的[74,75],如应用范围很广的 INTERNET,READ_PHONE_STATE 和 WRITE_SETTINGS 都是粗粒度权限,因为拥有这些权限的应用可以任意访问特定的资源.例如,超过 60%的 Android 应用程序需要 INTERNET 权限[44],拥有该权限的应用可以向所有的域发送 HTTP(S)请求,并连接任意目标地址和端口.

因此,恶意应用可以伪装成确实需要 Internet 访问的合法程序滥用 Internet 资源.

(3) 不充分的权限文档

Google 提供给开发者的权限文档不充分、不准确,有时还包含错误,容易造成开发者的失误,也会成为安全威胁的源泉(例如,申请过多不必要的权限).Felt 等人[33]发现:Android 2.2 文档仅列出了 78 个 method 的权限需求,事实上,应当说明 1 259 个 method 的权限需求,是原文档的 16 倍.此外,他们还发现了 Android 权限文档中的 6 处错误.

(4) 溢权问题

Android 权限机制难以发现和阻止应用程序申请过多的权限,产生所谓“溢权(overprivilege)”问题,突破了“最小特权原则”,是一种严重的安全隐患.研究表明[33]:多达三分之一的应用具有溢权问题,其中 56%的溢权应用有 1 个多余的权限,94%的溢权应用有 2~4 个多余的权限.总起来说,溢权问题共分两类:即恶意溢权和非故意溢权.

(5) 未提供“权限-API”对应关系映射集

Google 没有给出重要的“权限-API”对应关系映射集,Felt 等人[33]的研究弥补了这一空白.如图 8 所示,Android API[76]的架构由两部分组成:位于应用进程虚拟机中的 API 库和在系统进程中运行的 API 实现.API 调用有 3 个步骤:(i) 应用调用 API 库中的公开 API;(ii) 公开 API 调用 API 库中的私有接口(称为 RPC stub);(iii) RPC stub 初始化一个 RPC 请求,请求系统进程完成所需的操作.例如,某应用调用 *ClipboardManager.getText*(),则该调用被中继到 IClipboard$Stub$Proxy;然后,后者代理前者调用系统进程中的 ClipboardService 服务.Android 系统通过权限验证机制检查给定的应用程序是否具有指定的权限.权限验证机制在可信的系统进程中运行,并对 API 调用进行权限检查.如果 Android 应用程序没有申请权限就进行了 API 调用,则权限检查时就会抛出安全异常.Felt 等人[33]完成了“权限-API”对应关系映射集的构建,并在此基础上比较应用申请的权限和 API 调用,检查应用是否溢权.

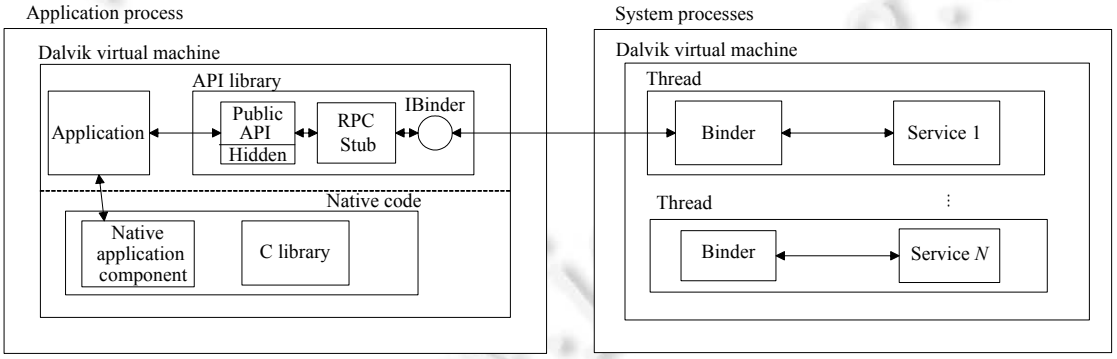

Fig.8 Android API calls and permission checks

图 8 Android 中的 API 调用与权限检查

2.4 Android的安全威胁

尽管 Android 系统精心设计了安全机制,但是由于 Android 的开放性以及在性能和安全性方面进行了折中,所以无法避免黑客的成功入侵.黑客通过利用系统漏洞,绕过系统安全机制、突破权限管理的“最小特权原则”

等方法,对 Android 系统形成不同类型和不同程度的安全威胁.根据 Android 的安全架构,其安全威胁来自不同的层次,如图 9 所示.

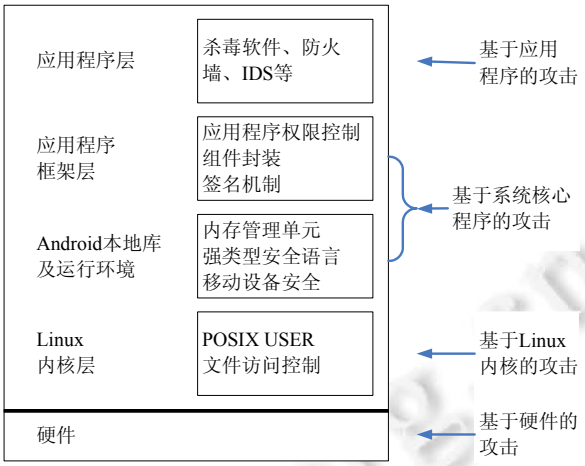

Fig.9 Security threats to Android

图 9 Android 面临的安全威胁

例如,来自 Linux 内核的攻击就会严重威胁 Android 系统的安全.Linux 每年都有近百个漏洞被 CVE 收录,通过内核漏洞利用获得系统最高权限,是黑客的首要攻击目标.

基于系统核心程序的攻击也是 Android 面临的主要威胁.例如,对 Intent 对象中目标组件的名称或动作进行修改,以劫持和伪造 Intent 等方式威胁 Android 安全.

基于应用程序的攻击更是五花八门.由于用户自行安装的应用程序来源多样,无法验证其安全性,使得针对应用程序的攻击成为 Android 系统最常见的安全威胁.

在实践中,安全威胁并不拘泥于单一的攻击形式,往往是多种攻击形式的相互配合,需从硬件到应用程序各个层次进行综合分析.

2.5 Google的应对措施

在日益增加的安全威胁下,Google 通过版本升级不断改进 Android 系统的安全性.例如:

- 借鉴 Windows 操作系统行之有效的经验[77,78],采取防止堆和栈缓冲区溢出攻击的措施,如:在 Android 2.3 版本之后增加了基于硬件的 NX(no eXecute)支持,不允许在堆和栈中执行代码.在 Android 4.0 之后,增加了“地址空间布局随机化(address space layout randomization,简称 ASLR)”功能,防止内存相关的攻击;
- 为防止恶意软件在用户未知晓的情况下提供高付费短信服务,Android 4.2[79]引入了附加的用户通知功能;
- 为防止前文提到过的“*Setuid*()-RLIMIT_NPROC 交互”型漏洞利用,Android 4.3[80]停止了对 *setuid*()/*setgid*()的支持;
- 为防止内核级提权攻击,Android 4.3[81]做出了一个重大变更,即,引入了具有强制访问控制机制的 SELinux 的支持.Android 4.4 进一步扩大了 SELinux 的支持.

3 Android 恶意软件

针对 Android 的恶意软件层出不穷,除熟知的木马、病毒、蠕虫、DDoS、劫持、僵尸、后门之外,近来又出现了如间谍软件(spyware)、恐吓软件(scareware)、勒索软件(ransomware)、广告软件(adware)和跟踪软件

(trackware)等恶意软件.特别地,如今越来越多的建议是将如影随形的广告软件列入恶意软件范畴.

3.1 Zimperlich

目前,一般将 root exploit 视为针对 Android 手机的专用术语(有时也称作 root Android),系指用户可以获取 Android 操作系统的超级用户权限,类似于 Apple iOS 的“越狱”.某些用户通过 root 越过手机制造商的限制进行设备定制:卸载预装在手机中的某些应用程序,或运行一些需要超级用户权限的应用程序.恶意软件则通过 Root 提升权限,绕过 Android 系统的安全机制.最著名的三大 Android root exploit 恶意软件是 Zimperlich[64], Exploid[82]和 RATC(RageAgainstTheCage)[83].许多其他恶意软件都是在它们的基础上研发的,如 DroidDream[84], Zhash[85],DroidKungFu[86]和 Basebridge[87].其中,DroidKungFu 以加密的方式同时包含 RATC 和 Exploid.当 DroidKungFu 运行时,首先解密和发起 root exploit 攻击,如果成功获取根权限,随后就可以为所欲为.

Android Zygote 是以 root 权限运行,并用于孵化所有 Android 应用的系统服务.Zygote 收到孵化 app 的请求后,对每个 app fork 一个子进程.在执行 app 代码之前,子进程通过 *setuid*()切换到分配给 app 的一个无特权的 UID.Dalvik VM 中的一段代码实现了上述过程.

在一般情形下,root 进程调用 *setuid*()不会失败,因此,Dalvik 虚拟机不检查 *setuid*()调用是否成功,导致 *setuid*()调用失败时不终止进程运行.Zimperlich[64]利用这个漏洞进行攻击的过程如下:

(1) 重复 fork 自身,最终耗尽所有的 PID,亦即达到用户进程数的上限 RLIMIT_NPROC;

(2) 向 Zygote 发送在新进程中孵化它的一个组件的请求.当 Zygote fork 一个子进程并试图创建 app UID 时,由于达到系统资源的极限,*setuid*()调用失败.这时,Dalvik 虚拟机并不异常终止,使 Zimperlich 代码得以与 Zygote 相同的 UID 运行,即,作为 root 进程运行;

(3) 重装可读写的系统分区,在其中创建 setuid-root shell.这样,Zimperlich 可在此后任何时刻均能获得具有 root 权限的访问.

Linux(以及 Unix)中 *setuid*()和 RLIMIT_NPROC 微妙的交互,是许多类似安全问题(统称为 *Setuid*()-nproc 极限型漏洞)的根源,例如,RATC 恶意软件也是利用这种漏洞成功地攻陷了 Android 调试桥接守护进程(Android debug bridge daemon,简称 ADBD),从而取得根权限.因此,对此问题如何改进 Linux 的内核,成为近期研究者讨论的热门问题[88].

3.2 Android恶意软件分类

Android 恶意软件有不同的分类方法,按照传统的方法可以如下分类.

- 木马类.这是一类多发性恶意软件,较著名的有 Zsone[89],FakeNetflix[90],Fakeplayer[91],Spitmo 和 Zitmo[92]等;
- 病毒类.例如,Obad[93]是一款有名的蓝牙蠕虫病毒;
- 后门类.这也是一类多发性的恶意软件,较著名的有 Obad[93],RATC[83],Basebridge 和 KMin[3]等.注意,Obad 既是病毒也是后门;
- 僵尸类.文献[3]中给出了一些僵尸的例子:Geinimi,Anserverbot 和 Beanbot.此外,NotCompatible[1]也是一款著名的僵尸类恶意软件;
- 间谍软件类.例如,GPSSpy[3]和 Nickyspy[89];
- 恐吓软件类.例如,Koler[1];
- 勒索软件类.例如,FakeDefender.B[94].注意,勒索软件往往同时也是恐吓软件;
- 广告软件类.例如,Uapush.A[1]是一种著名的广告软件木马;
- 跟踪软件类.例如,SMSTracker[1]是一款著名的跟踪软件.

Zhou 等人[3]根据 Android 恶意软件的特征进行如下分类.

(1) 恶意软件安装(malware installation)

- 重打包(repackaging);

- 更新攻击(update attack);
- 诱惑下载(drive-by download);
- 其他.

(2) 恶意软件运行(activation)

(3) 恶意载荷(malicious payloads)

- 提权攻击(privilege escalation);
- 远程控制(remote control);
- 付费(financial charge);
- 信息收集(information collection).

(4) 权限使用(permission uses)

他们的分析指出:在其 1 260 个恶意软件样本中,有 1 083 个属于重打包型恶意软件,占比为 86%.

3.3 提权攻击

通过权限提升实现成功攻击,是黑客最常用的手段之一.有别于前面阐述过的 root exploit 型攻击,这里所说的提权攻击系指没有任何权限的恶意程序能够通过第三方程序获得所需的权限.Davi 等人[95]关于这类提权攻击给出了下述定义:“一个拥有较低(较少)权限的应用程序访问拥有较高(较多)权限的组件而不受任何限制,称为 Android 系统的提权攻击”.有的文献也称此类提权攻击为“权限再次授权(permission re-delegation)”.

如图 10 所示,app3 中的组件 1 受权限 P1 保护,app1 中的组件 1 没有权限,app2 中的组件 1 拥有权限 P1.因此,app1 中的组件 1 不能直接访问 app3 的组件 1,而 app2 中的组件 1 可以访问.但 app2 中的组件 1 不受任何权限保护,故 app1 中的组件 1 可以访问它,然后,通过它获得访问 app3 中组件 1 的资格.这样,app1 中的组件 1 就绕过了 app3 中组件 1 的保护机制完成了提权攻击.

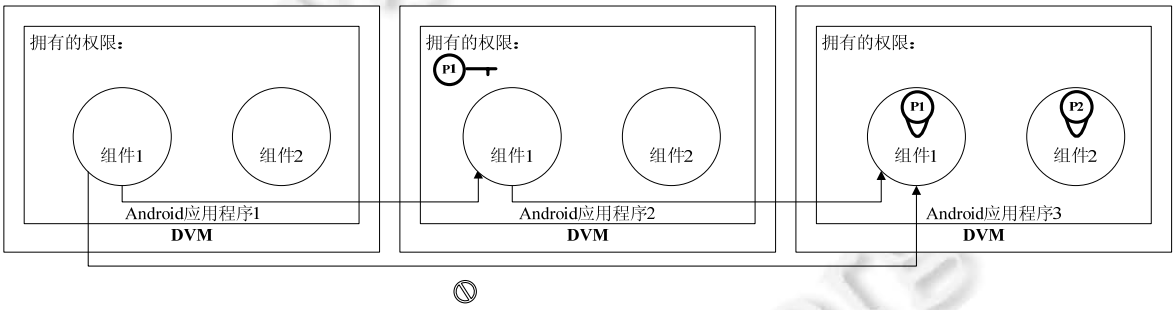

Fig.10 Privilege escalation attack on Android

图 10 Android 提权攻击

具体的提权攻击实现方法[96−99]各不相同,有的利用 Intent 盗窃/劫持[99],有的需要多个应用程序之间的相互配合等.

有些文献将提权攻击分为两大类[32,98]:困惑代理(confused deputy)攻击与合谋攻击(collusion attack).1988 年,Hardy[100]提出了所谓“困惑代理”的概念,问题源于操作系统编译程序的“一仆二主”,即,编译程序运行时同时具有分别来自两个调用的不同权限.对于智能手机,具有不同权限集的应用可以彼此通信,使恶意应用具有非法使用某些正常 app 权限的可能性.典型的攻击场景是:当一个有权限(但“困惑”)的 app 接口有漏洞时,恶意软件利用该接口漏洞实施困惑代理攻击.

合谋攻击需要多个应用程序之间的相互配合,又可细分为直接攻击[101]与间接攻击[102]两个子类.顾名思义,前者依靠应用程序之间的直接通信;而后者依靠它们之间的间接通信.图 11 是一种合谋攻击的例子,首先,攻击源 A 通过代理组件(C_1,C_2,…)到达泄露点 P 种下种子(植入恶意代码等);然后,通过攻击源 B 二次对泄露点 P 进

行攻击,进行提权操作或数据窃取等.这样,即使检测器可以检测到攻击源B的攻击,攻击源A却依旧难以被发现.

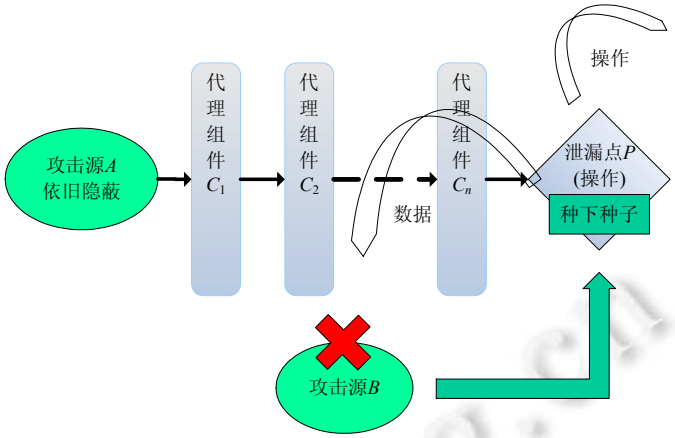

Fig.11 An example of collusion attack

图 11 合谋攻击举例

3.4 中间人攻击MITM

中间人(man-in-the-middle,简称 MITM)攻击是不同领域中随处可见的攻击方法,在 Android 中,主要针对通过 SSL/TLS 加密的 API.众多学者[103−105]对此进行了深入的分析,有些文献[106,107]给出了具体的对策.

Android 应用程序通过 SSL/TLS 加密的 API 安全地传输敏感信息,但 app 开发人员通常自行实现标准的 SSL/TLS 证书验证程序.许多这类定制的实现故意接受所有自签名的证书,有的甚至不作任何校验便接受所有的证书,有的包含各类安全漏洞,使 MITM 攻击有了成功的机会.

Fahl 等人[103]实现了一个名为 Malldroid 的工具,对 13 500 个包含 SSL/TLS 代码的 app 进行了静态分析,发现其中 1 074 个(占比为 8.0%)app 含有易受 MITM 攻击的安全漏洞.他们在这些含有漏洞的 app 中选择了 100 个热门应用进一步作了人工分析,发现其中 41 个应用程序的确不能对 MITM 攻击免疫.文献[105]对 Android 官方应用市场 Google Play 中的 11 748 个 app 进行的自动分析表明,其中,10 327 个(占比为 88%)app 使用了加密 API 并至少犯了一个错误.文献[104]的分析指出:在许多与安全相关的热门应用与库中,SSL 证书认证已经完全失效,某些流行的第三方库(如 ACRA)用有漏洞的实现覆盖了内置的 SSL 证书认证模块.

3.5 恶意软件变形技术

为规避恶意软件检测器和分类器的自动检测,恶意软件采用了一系列自保护的变形(transformation)技术.有的文献,如文献[102],也称其为隐身(stealth)技术,包括熟知的混淆、代码加密、密钥置换、动态加载、代码反射和本地代码执行等.从下一节的分析可以看出,这些措施对于反制商业反恶意软件的侦测还是十分有效的.

Rastogi 等人[19]构建了一个包含各类恶意软件变形技术的系统框架 DroidChameleon,对变形方法进行了分析.他们将变形分为以下 3 类:平凡的变形、通过静态分析可能发现的变形以及只有通过动态分析才能发现的变形.其中,平凡的变形不需要进行代码级的改变或 AndroidManifest 中存储的元数据的改变,例如重打包、反汇编与再汇编;通过静态分析可能发现的变形包括重命名包、重命名标识符、数据加密、代码重排序、无用码插入、组合变形技术等;只有通过动态分析才能发现的变形包括代码反射与字节码加密.

文献[108]指出:许多反恶意软件使用已知恶意软件的平凡特征,例如包名、类名和方法名作为检测特征,所以,一些平凡变形仍然可以逃避基于特征的反恶意软件检测.

3.6 商业反恶意软件健壮性评估

许多研究者对主流商业反恶意软件的健壮性进行了评估.

Zhou 等人[3]通过一年多的积累,采集了从 2010 年 8 月~2011 年 10 月的 49 个不同族中的 1 260 个 Android 恶意软件样本,经过对它们的安装与运行特征进行了系统分析之后得出以下结论:(1) 86.0%的恶意软件重打包合法的 app,增加恶意载荷;(2) 36.7%的恶意软件通过平台级的漏洞利用进行提权攻击;(3) 93.0%的恶意软件具有使用户手机成为僵尸的能力.令人遗憾的是,他们选择了 4 个有代表性的移动反病毒软件:AVG(AVG antivirus free v2.9),Lookout(lookout security & antivirus v6.9),Norton(norton mobile security lite v2.5.0.379)和 TrendMicro (TrendMicro mobile security personal edition v2.0.0.1294),它们在最佳和最坏情形下只能分别检测出 79.6%和 20.2%的恶意软件.结论是:商业反恶意软件不够健壮,亟需下一代反移动恶意软件的解决方案.

Rastogi 等人[19]通过他们构建的框架 DroidChameleon 对下面 10 个有代表性的 Android 反恶意软件(括弧外是公司名称,括弧内是软件名称)进行了系统的分析:AVG(antivirus free),Symantec(norton mobile security), Lookout(lookout mobile security),ESET(ESET mobile security),Dr. Web(Dr. Web anti-virus light),Kaspersky (Kaspersky mobile security),Trend micro(mobile security personal Ed.),ESTSoft(ALYac Android),Zoner(zoner antivirus free),Webroot(Webroot security & antivirus),也得出了相似的结论:商业反恶意软件亟待提高对抗恶意软件变形技术的能力.

Zheng 等人[20]也研究了 Android 反恶意软件的健壮性,他们构造了一个自动和可扩展的平台 ADAM,对商业反病毒系统进行了压力测试.但他们只实现了文献[19]中恶意软件变形的一个子集.

4 Android 安全分析

Android 安全分析基本上可以分为静态分析和动态分析两类.也有研究者将 Android 安全分析分为基于特征与基于机器学习的分析两类.

4.1 静态分析

静态分析在安装前检测 Android 应用程序中的安全漏洞,通常的分析步骤是:下载 Android 系统的应用程序安装包(Android application package file,简称 APK),通过如 Ded[109]和 Dex2Jar[110]之类的工具进行反编译,将 APK 文件反编译为 Java 源文件或 JAR 文件,然后对其进行源代码级的解析.静态分析的优点是无需运行代码,无需像动态分析那样改写 Android 系统源码、要求用户对 Android 系统进行重定制和安装定制版的 ROM,因此,速度快,轻量级.静态分析的缺点是:因为无法真实模拟程序的动态运行,所以存在误报问题.

基于权限的检测工具 Kirin[111]是早期有代表性的静态分析方法之一,它通过定义的一组规则识别危险的权限组合,其缺点是误报率较高.其他的静态分析解决方案还有 SCanDroid[112],DroidChecker[113],Chex[114], AndroidLeaks[115],Epicc[116]等,但这些工具仍然有其局限性:有的未进行组件间通信分析,有的未考虑 Android 的事件驱动编程范式.CoChecker[117]对此进行了改进,提出基于 Android API 调用将泄露路径分为两类的方法:权能泄露路径和敏感数据泄露路径.CoChecker 可以自动识别 Android 应用中的这两类泄露路径,因为两种泄露路径都可能跨越多个组件,因此,进行 ICC 分析是一项基本任务.

静态分析的方法多种多样,例如,ScanDal[118]以 Dalvik 虚拟机字节码为输入,检测 Android 应用程序的私密信息泄露.Chex[114]是基于组件的静态分析方法.SCanDroid[112]使用数据流分析方法进行静态分析,从应用的 AndroidManifest.xml 文件中提取权限,然后自动检测应用的数据流是否与这些权限一致.DroidChecker[113]使用控制流图搜索和脏数据分析方法自动分析可能的敏感数据泄露路径.

机器学习方法也可以用于静态分析,如文献[14,53]等.

4.2 动态分析

相对于轻量级的静态分析,动态分析则是重量级的程序运行时的分析.一般情形下,需对 Android 系统进行重新定制与改写,包括改写安全机制;在原生 Android 系统中加入监视器,实时监视数据的流向;在危险函数调用时,检测所需权限等.动态分析的优点是检测精度较高,缺点是重量级:需要修改 Android 系统源码,形成用户全新定制的操作系统,同时需要通过更新 ROM 安装这种定制系统.早期有代表性的动态分析工具如 Stowaway[33,119],

重新定制 Android 系统的工具如 Quire[120].Felt 等人[33]分析了 940 个 Android 应用程序,发现其中三分之一存在突破最小特权原则的安全威胁.

TaintDroid[121]提出了一种动态脏数据检测技术:对敏感数据进行污点标记,当这些数据通过程序变量、文件和进程间通信等途径扩散时进行跟踪审查.如果污点数据在一个泄露点(如网络接口)离开系统,TaintDroid 就在日志中记录数据标记、传输数据的应用程序和数据流向目的地,实现对多种敏感数据泄露源点的追踪.但 TaintDroid 只考虑数据流,未对控制流(程序间交互、组件间交互、进程间交互)进行分析.TaintDroid,Asdroid[122]和 Crowdroid[123]等都是一类基于行为的动态检测方法.

XManDroid[97]是一种基于系统策略的动态分析方法,它扩展了 Android 应用架构,由 3 个模块组成:运行监控器、应用安装器和系统策略安装器.运行监控器是 XManDroid 的核心,由访问监控器、决策器、系统视图、权限、系统策略和决策组件构成.其中,新增组件的功能如下:决策器判断是否建立新的组件间通信链接;系统视图组件维护运行系统的状态;系统策略组件是系统策略规则数据库;决策组件是存储决策的数据库.应用安装器包括 Android 标准组件 Package Manager 和 Permissions 以及新增的系统视图组件.由于安装新的应用程序会改变系统状态,故扩展了 Package Manager 的功能,使其可与系统视图进行通信.系统策略安装器是新增模块,由策略安装器、系统策略、决策和系统视图这 4 个新组件构成,提供在 Android 中间件层增加或更新系统策略的功能.XMandroid 通过系统策略检查算法判断能否建立新的组件间通信链接,有效地防止了运行时的权限提升.XMandroid 的局限性是:(i) 未考虑通过隐蔽通道实现的权限提升;(ii) 仅考虑了 ICC,但应用程序间的通信还包括标准的 Linux 进程间通信机制等;(iii) 未考虑其他公开信道,如互联网 Socket 或文件系统通信等.

机器学习方法也可用于动态分析,如文献[51,59]等.

4.3 机器学习分析方法

近年来,许多研究者尝试借助机器学习方法[41,44,51−61,123−126]进行恶意软件的分类与检测.用机器学习方法构成的分类器可以模拟 Android 应用的行为,区分良性与恶意软件.分类器通过分析应用的静态或动态行为特征做出判断,静态特征可以通过反编译 Android Dalvik 字节码获得的结果进行抽取;动态特征可以通过观察程序运行时的行为进行搜集.通过 Android 权限、代码或运行,可以抽取到各种不同类型的特征.

采用什么样的机器学习方法是 Android 分类器是否有效的关键.另外一些重要因素是,提取什么特征、如何提取特征以及提取多少特征等问题.目前,这些问题都还没有一个完美的答案.在 Android 中,常用的机器学习方法包括 *k*-Means、逻辑回归(logistic regression)、*K*-最近邻(*K*-nearest-neighbor,简称 KNN)、直方图(histogram)、决策树(decision tree)、贝叶斯网络(Bayesian network)、朴素贝叶斯(naïve Bayes)、随机树(random tree)、随机森林(random forest)、支持向量机(support vector machine,简称 SVM)等.

如何度量 Android 分类器是否有效呢?4 个基本指标是:真阳性(true positive,简称 TP),表示正确分类为恶意软件的恶意样本个数;假阳性(false positive,简称 FP),表示错误分类为恶意软件的良性样本个数;真阴性(true negative,简称 TN),表示正确分类为良性软件的良性样本个数;假阴性(false negative,简称 FN),表示错误分类为良性软件的恶意样本个数.由此可以构成下面的一些常用指标:

- 检测率(detection rate),亦即真阳性率 $TPR=TP/(TP+FN)$,有时也称为召回率(recall rate),表示所有恶意样本中正确分类为恶意软件的比例;
- 误报率(false alarm rate),亦即假阳性率 $FPR=FP/(FP+TN)$,表示所有正常样本中错误分类为恶意软件的比例;
- 漏报率(missed detection rate),亦即假阴性率 $FNR=FN/(TP+FN)$,表示所有恶意样本中错误分类为良性软件的比例;
- 分类精度(accuracy,简称 ACC),是所有正确分类的样本与所有样本之比,即:

$$ACC=(TP+TN)/(TP+TN+FP+FN).$$

此外,经常与召回率一起讨论的指标有正确率(precision),系指真正的恶意软件在分类为恶意软件中的比例,即:正确率=$TP/(TP+FP)$.