立体安全防御系统TDSD-Droid的实现

刘 洋 邵旭东 潘程达 胡正梁
(公安部第三研究所信息安全技术部 上海 201204)

摘 要 随着智能终端的日益普及,便捷易用的Android操作系统得到了广泛的使用。标准的Android安全架构ASF缺乏强有力的保护机制,而现有和正在研究的标准的Android安全加固技术都有一定片面性。TDSD-Droid通过吸收SELinux及其它Android安全加固技术优点,引入了基于内核的MAC机制;创新性地运用Flask访问架构实现了新的MMAC机制;创新性地实现了柔性安全策略FSP适配机制;创新性地实现了安全策略学习机制;同时基于TF智能卡实现了TDSD-Droid的完整性验证功能。TDSD-Droid为Android终端实现了一个上下一致、前后呼应的立体安全防御系统。

关键词 安卓,安全加固,强制访问控制,自主访问控制
中图法分类号 TP391.8 **文献标识码** A

Implementation of Three-dimensional Security Defense System

LIU Yang SHAO Xu-dong PAN Cheng-da HU Zheng-Liang
(Department of Information Security, The Third Research Institute of Ministry of Public Security, Shanghai 201204, China)

Abstract With the increasing popularity of intelligent terminals, convenient-to-use Android operating system has been widely used. The standard Android Security Framework is lack of strong protection mechanism, even the existing and developing security technology for standard Android are one-sided. TDSD-Droid adopted the advantages of SELinux security enhancement and other Android security technology, and implemented a MAC mechanism in kernel, a new MMAC mechanism based on Flask access architecture, a novel Flexible Security Policy adaptation mechanism, an innovative security policy learning mechanism, and a new integrity verification function based on TF smart card. It achieved a consistent three-dimensional security defense system for Android terminals from top to bottom.

Keywords Android, Security enhancement, Mandatory access control, Discretionary access control

1 简介

智能手机与智能平板等智能终端日益普及,它们已深深融入到了日常生活与工作中。这些智能终端中封装了智能操作系统,向用户提供强大的功能。据统计,目前智能终端中大多数部署的是Google的Android操作系统;该操作系统所占有的比率还在不断上升[1]。

智能终端功能强大,便捷易用,越来越多的用户用它来存放与处理个人和公司的敏感数据。但是这些嵌入式系统上运行的操作系统大都缺乏强有力的保护机制,使得这些敏感数据,甚至设备本身极易被破坏和盗取[2]。Android也不例外。尽管标准Android缺省引入了Android安全架构ASF(Android Security Framework)[3],但Android智能终端仍然薄弱不堪,常被不良或恶意程序侵入,进而获取root的权限、盗取用户敏感数据等。

正是看到这一点,许多研究机构与相关企业纷纷着手相关的安全加固研究。基于Android 4.1.2以及自主设计的三防手机HD4,本文展现了立体安全防御系统TDSD-Droid(Three-Dimensional Security Defense-Droid)的设计与实现。同时,随着这些安全加固技术在Android上的实现与成熟,它将有力推动这些技术在其它智能终端操作系统上的移植。

已有的或正在研究的安全加固技术,基本落实在访问控制的扩展与加固上。它们有的立足于应用程序,有的则立足于数据;有的着重于进程间隔离,有的着重于sandbox分层阻挡技术。像Saint就让应用程序开发者可以将开发应用程序的接口保护起来,从而较好地实现不同应用程序的隔离;而TrustDroid则在Linux内核引入MAC访问控制机制,对进程想要访问的资源给予更精细粒度的权限划分[4]。这些技术都对Android的安全加固做了有意义的尝试,但它们都有一定的片面性,不能给予Android操作系统及智能终端用户提供立体全面的安全防御。

通过对标准Android ASF安全架构等安全加固技术的研究,对Android移动智能操作系统进行加固改造,TDSD-Droid实现了一个有效的、可行的立体安全防御机制,为智能终端提供了一个安全可靠的运行环境。本文将展现其主要工作:

刘 洋(1973—),硕士,主要研究方向为信息安全、操作系统安全加固和嵌入式系统,E-mail:liu_yang182@163.com;邵旭东(1976—),男,硕士,助理研究员,主要研究方向为信息安全、入侵检测与防御;潘程达(1980—),男,硕士,主要研究方向为信息安全;胡正梁(1981—),男,硕士,主要研究方向为信息安全。

1)建立一个立体的安全加固架构:基于 TE 实现 Linux 内核层的强制访问控制 MAC(mandatory access control)[5]访问机制;基于 TE 实现用户空间层强制访问控制 MMAC (Middle MAC)访问机制;

2)根据最终用户、应用开发人员以及系统开发人员的不同使用场景,定义了几套不同参考安全策略,实现了柔性安全策略 FSP(Flexible Security Policy)适配机制;

3)建立了在线安全策略自动化学习机制,学习 Android 终端用户的使用习惯,吸收合理安全策略,保证参考安全策略的先进性。

4)实现确保核心模块不被篡改或替换的完整性验证功能。

本文将描述 TDSD-Droid 的设计与实现;将列举面对利用 Android 安全漏洞的攻击,TDSD-Droid 有效防御的攻防实例;还将评估 TDSD-Droid 的有效性和可用性。

2 背景

本节将先展现标准 Android 的安全框架 ASF,接着讲述 SELinux 的安全加固原理,最后会分析当前主流的 Android 加固技术。

2.1 ASF

Android 是基于 Linux 内核的 Java 虚拟机系统。与一般 Java 虚拟机不同的是,其上的 Java 应用程序是以进程的形式运行,而不是以线程。该系统上本身集成有一安全架构 ASF。该架构由 3 部分组成:Linux 自主访问控制 DAC(Discretionary Access Control)机制、JVM 的 sandbox 机制以及 Android 许可机制。

在 ASF 中,Android 许可机制是 Android 专有的。它基于组件间通信机制 ICC(Inter-Component Communication)[6],以及 Android 专有驱动 Binder 实现的轻量级进程间通信机制,对某应用想访问的系统资源,或其它应用资源进行访问许可判决。应用程序要在其配置文件或程序中显式申请要访问系统资源的许可;该许可判决发生在应用程序安装时,此时会请求用户,核准该许可申请;否则该程序将得不到相应资源的访问权。在这个许可模型中,一方面许可要由一般用户判决,这让他们极为难;另一方面,该许可模块只是对程序接口或资源进行禁止与使能的粗放型管理;最后,其裁决是在安装时有效,完全不能适应运行时的变化及运行时保护需求。

其次,一般 Android APK 应用程序以进程的形式运行,因此较基于线程虚拟机具有更强的隔离。但是出于执行效率等方面的考虑,Android 又提供了 NDK(Native Development Kit)或 JNI(Java Native Interface),对底层 Linux 内核进行直接调用,比如 fork(),直接将这种隔离与 Java 虚拟机的 sandbox 保护旁路了。所以,ASF 中 Java 虚拟机的安全保护机制,对于不良的程序形同虚设。同时 Android 的 Installd 等 Daemons 后台服务程序,就是直接运行在 Linux 内核上的进程。所以即使限制 Android Java 程序运行在 Dalvik 虚拟机中,不良程序也还可以利用这些 Daemons 发起攻击。

最后,基于 DAC 访问控制机制[7],Linux 会根据进程生成的 UID 与 GID,以及文件等资源所属的 UID 和 GID,来判断某进程是否拥有某资源的访问权。这种访问控制机制存在以下先天缺陷:1)一旦 A 进程拥有 B 进程的 UID 与 GID,则 A 进程就拥有 B 进程的所有权限;2)该访问控制机制对 root 用户无效,不良程序可以通过 root 所有的具有 setuid 位的程序来获得 root 用户权限,进而实施其为所欲为的破坏;3)该访问控制只在资源第一次被访问有效,在其它的运行时的访问时并无效;4)该访问控制对资源权限的划分不够精细,达不到针对不良程序进行更精细化防护的要求。因此 SELinux 等纷纷在操作系统中引入了 MAC 访问机制。

2.2 SELinux

SELinux(Security Enhanced Linux)是 Flask 安全模型[8]的一个实现,它基于安全策略实现了 MAC(mandatory access control)访问控制机制。在 SELinux 中的安全服务器 SS(Security Server)与资源管理器两个组件,分别负责基于安全策略的访问判决和基于安全策略的访问实施。通过该 MAC 机制有效地保护了底层的文件、IPC(Inter-Process Communication)和内存等资源。

在 SELinux 的 MAC 机制中,支持多种访问控制政策:基于角色访问控制 RBAC、分层安全控制 MLS 和类型安全加固 TE(Type Enforcement)等。其中 TE 是最基础的 MAC 技术:在这种安全加固技术中,访问被分为主体(就是运行着的进程)和客体(就是进程要访问的某文件等资源);所有的主体与客体都被标记了相关的安全上下文。下面就是一条访问策略语句:

allow TSub Tobj:Cobj Opers

其中,Tsub 是主体进程的类型,Tobj 是被访问目标客体的类型,Cobj 是客体的类别,而 Opers 则是主体对客体访问操作的集合。在这条语句中,Cobj 决定了 Tobj 所属的类,以及对 Tobj 可实施的操作。如果某 TSub 想对某 Tobj 访问操作,则请求的操作必须在上述策略语句的 Opers 集合中;否则该访问将被拒绝。

另外,SELinux 还实现了用户空间客体管理器 USOMs (UserSpace Object Managers),以将 MAC 访问控制机制扩展到用户空间的进程,有效地实现用户空间相关数据资源的精细化管理。GConf 就是一个 USOMs 的具体实现[9]。

SELinux 已广泛应用于各类安全服务器,辅助于其它安全措施,已被验证为一般计算机系统提供了有效的安全保护。虽然 Android 的底层使用的也是 Linux,但想将 SELinux 迁移其上,由于 Android 文件系统、Binder、Dalvik 等的独特性,也不是件容易的事[10]。

2.3 关于 Android 的安全加固改良

正是看到 MAC 机制安全防护上的优势,各机构对 Android 安全加固也纷纷引入 MAC 技术。主流的安全加固改良产品有:Saint[11]、TrustDroid[12]、TaintDroid[13]、SEAndroid 与 XManDroid[14]等。虽然它们都采用了 MAC 机制,但在具体的实现细节上,各有侧重与不同:有的着重于内核加固,有的则着重于用户空间层加固;有的基于 TOMOYO Linux 实现加固,有的则基于 SELinux 实现加固;有的从 Flask 访问架构来改良 MMAC,有的则利用标准 Android 许可机制来改良 MMAC。

这些 Android 安全加固都有一定的不足。其中 SEAndroid 的 MLS(MultiLevel Security)MAC 政策因为缺少

RBAC(Role Based Access Control)机制的配合，难以提供基于敏感级别的有效防护；同时其所实现的 MMAC 也仅是针对标准 Android 许可机制的改良，无法针对用户空间的资源实施运行时的保护。而 Saint、TaintDroid 与 XManDroid 根本没有实现基于内核层的 MAC；TrustDroid 虽然是基于内核层 MAC 加固，但完全没有考虑针对用户空间资源的保护。

最后，这些安全加固实现一般都会提供参考安全策略，用作强制访问机制判决主体对客体访问许可的根据。但没有谁证明所提供的安全策略就完全合理，或随着应用的增长等环境的变化，这套策略仍是合理的。而且对所做 MAC 加固模块都未进行完整性验证，不能防止不良程序对这些关键软件模块篡改或替换。

3 安全加固需求分析

3.1 针对 Android 的攻击

就目前 Android 的攻击而言，有的来自本地第三方应用，也有的来自 Internet 网；而这些攻击的目标则涵盖 Android 的各个软件层次：有的是针对底层内核的攻击，有的是针对中间层 server 的攻击，还有的是针对上层应用的攻击。从攻击的手段看，攻击的形式主要有：

1)root 攻击：利用 root 权限不受限展开的攻击。

2)窃取攻击：通过不良代码，窃取终端用户的数据。

3)暗通道攻击：通过收集终端的行为数据并分析，进而获得用户的私人信息。

4)内外勾结攻击：内部人员给不法人员或不良程序开放“绿色通道”的攻击。

现在各类 Android 攻击中，用得较多的就是利用 root 权限不受限的特点，利用标准 Android 的漏洞，获取 root 权限，进而绕过标准 Android 的 ASF，恣意对 Android 的内核、中间层，及各类应用展开攻击。

3.2 基于 Android 要做的安全加固工作

正如前面提到的各 Android 安全加固技术，面对利用 root 权限及针对内核资源客体的攻击，应引入 SELinux 基于内核层的 MAC 机制。因为该机制能有效防止利用 root 权限不受限的攻击，同时可以给予 Linux 内核层的各类资源更精细的权限划分与保护。

但是基于内核 MAC 访问控制保护也有缺憾，不能对上层资源进行针对性保护。比如，Camera 设备在底层受到了内核 MAC 的保护，可以对 Camera 这个设备的打开、关闭、IOCTL、属性查找等访问操作实施保护，但对 Camera 服务提供的拍照与录相不能提供有效的保护；换句话讲，上层应用一旦获得了 Camera 设备及相关文件的访问权限，就拥有了 Camera 服务这个中间件的相关权限，结果只需拍照功能的程序也拥有了摄像权限。因此，还有必要在用户空间也提供类似内核底层的 MAC 保护机制，以更精细化地保护上层资源。

当然，有了用户空间的 MMAC，并不意味着可以省去底层内核 MAC。因为没有底层内核的安全加固，所有上层的任何安全加固措施，面对内核的漏洞和不良程序，就像是建立在沙堆上的楼阁。所以，关于 Android 合理的加固，应是 MMAC 与 MAC 的有机结合：主体访问客体，当牵涉到用户层资源，则要基于 MMAC 实施许可判决；当牵涉到内核层资源，则要接收 MAC 的访问许可判决；只有在两者都许可的情况下，访问才会成功。

MMAC 与 MAC 的最终实现，离不开安全策略的制定。而面对 Android 终端的用户、应用开发者、安全管理员等，应该有不同的安全权限；针对这些不同角色，Android 终端应工作于不同的安全场景。因此，在安全加固中，需要支持基于角色的访问控制 RBAC；同时针对上、下两层强制访问控制机制，应保持 RBAC 一致性；而且一般用户应对这套角色识别机制透明。除了根据用户定制不同的安全策略，还应根据时间、地点等实际运用的不同，为 Android 用户提供灵活的安全策略配置。

安全策略的合理性或正确性难以验证。策略制定过紧，会导致终端用户得不到正常的响应；而策略制定过宽，则会让不良程序有机可趁。安全策略只有向广大终端安全管理员学习，汇集来自具有实战经验的安全策略，并经过提炼才是最好的。因此作为 MMAC 或 MAC 的辅助，安全策略机制应是学习型的：基于后台服务程序，通过学习与提炼安全策略，生成最合理的参考策略。

尽管前面 MMAC、MAC、安全策略动态配置机制、安全策略学习型机制，已为 Android 终端提供了较完备保护，但面对这些保护组件或软件自身被篡改或替换的攻击时，则显得力不从心。因此，还应辅以完整性验证机制，来验证确保这些安全软件的正确与合法性。

事实上，针对 Android 实现安全加固，应该建立一个完备且灵活的立体防御体系。唯有此，才能有效地应对上节中所述的各类攻击。

4 TDSD-Droid 实现

在这节中，将描述 TDSD-Droid 的实现。TDSD-Droid 引入了 MAC，实现了 MMAC 与安全策略灵活配置机制；同时，基于远程后台上实现了学习型安全策略服务，基于 TF 智能卡实现了完整性验证算法。

4.1 MMAC 及其它终端侧安全加固技术

4.1.1 综述

TDSD-Droid 向 SELinux 及其它 Android 安全加固技术学习，基于 TE 实现了内核底层的 MAC。同时基于 Flask 访问架构，在 TDSD-Droid 中构建了一个新的 MMAC。同时支持针对 MMAC 与 MAC 安全策略的灵活配置。关于内核层 MAC 参考了 SEAndroid 的实现，这里不展开讨论。TDSD-Droid 实现框架如图 1 所示。

从图 1 可以看到，MMAC 中的资源管理器，其功能代码散落在了各处：包管理器 PM(Package Manager)、服务管理器 SM(Service Manager)和内容提供器 CP(ContentProviders)。当有主体(例如运行的应用程序)想要访问某服务、接收某 Intents 或读写某 ContentProvider 的数据时，MMAC 的安全服务器 USSS(User Space Security Server)就在标准 Android 访问流程中插入了相应的判决流程。

TDSD-Droid 除了实现 MMAC 基于 TE 的访问控制逻辑，还实现了 FSP，以便吻合 Android 中间层软件的访问控制的需要，尤其是要吻合 Android 终端由于角色等运用场景变化，而灵活地切换 Android 终端的安全上下文。同时，在这个安全上下文的切换中，实现了与底层联动，确保了上下安全工作环境一致，前后的安全策略之间不冲突或矛盾。

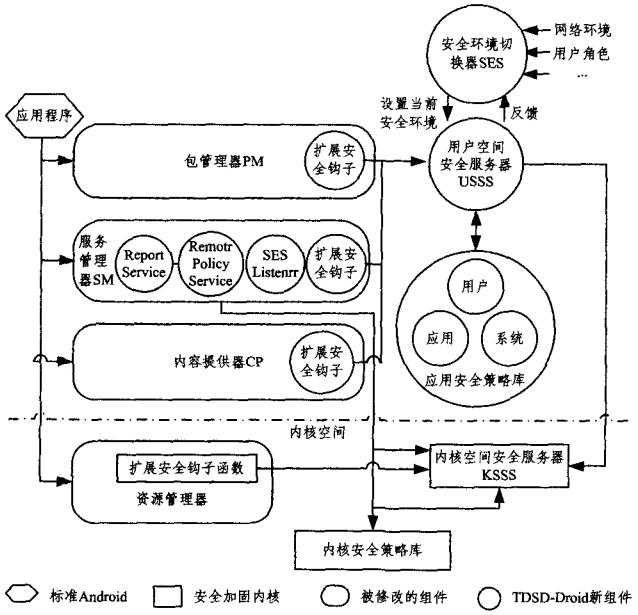

图 1 TDSD-Droid 架构

4.1.2 MMAC 实现

MMAC 中 USSS 处于核心位置，通过它来实现上层主体对属于上层的资源访问许可判决。通过 USSS 和 KSSS(Kernel Space Security Server)，清晰地划分了用户空间与内核空间的安全问题。同时 USSS 的引入，为用户空间使用更灵活的安全策略提供了机会。

USSS 所实现的访问控制也是基于 TE 政策，将根据主体的类型、客体的类型、客体关联类别和客体向主体许可操作集，来判决主体对上层客体资源的访问是否被许可。

USSS 被实现为一个 Android 系统 service，驻留在 Android 的服务管理器 SystemServer 中。它向 USOMs 提供了一个接口，以便主体对某客体访问时，通过 ICC 来申请 USSS 的访问许可判决。USSS 的工作机制与内核层的 KSSS 工作机制大体一样。

TDSD-Droid 的 USOMs 是通过改造标准 Android 的相关组件来实现的，因此处于 Android 中间的服务、应用都可以是个 USOM。预先集成在 Android 映像中的服务与 App 应用，都扩展了安全钩子函数，以获得 MMAC 的保护，确保各自拥有资源客体的安全。而对于后面动态安装的 service 或 App 则分别通过 System Server 和 Package Manager 提供的接口，选择是否要 USOM 扩展函数。如果选择扩展安全钩子函数，则会根据这些程序的 UIDs，分配缺省的安全访问控制函数。当然也可以不选择扩展；这样做的好处是：不扩展的程序将得以像在标准 Android 上一样运行。但若选择不扩展安全钩子函数，这些程序就将得不到 MMAC 的安全防护。表 1 显示了在 TDSD-Droid 中实现的参考 USOMs，以及这些资源管理器所能提供的安全加固操作。

表 1 中，PackageManager 负责 App 程序包的安装与卸载，并负责在 Android 运行时，选定合适的组件来完成指定的任务。作为一个 USOM，PackageManage 基于标准 Android 做了相应的扩展：适配用户空间 App Type 与内核空间 domain Type；指定任务执行主体安全加固条件量。对于初始就集成在 Android 系统中的 App 应用程序，在终端启动时就会打上各自的安全上下文标记；而对于第三方应用，一般难有吻合 MAC TE 的安全上下文。通过 UIDs，Android 中安装运行的第三方应用归为 untrust_app 或 system_app 等 App Type，赋予这些 App 应用基本的安全属性。同时，通过在针对指派任务的执行组件的查询中加入组件安全属性判决条件，使得具体上层任务启动也在 USSS 保护下。

表 1 TDSD-Droid 已实现的 USOMs

USOM	安全加固的功能函数
PackageManager	findPreferredActivity, getInstalledApplications, getPackageInfo, installPackage
ActivityManagerService	grantURIPermission, moveTask, registerBroadcastReceiver, sendBroadcast, startActivity
AudioService	setStreamVolume, setVibrateSetting
PowerManagerService	acquireWakeLock, isScreenOn, preventScreenOn, reboot
SensorManager	getDefaultSensor, getSensorList
LocationMangerService	addProximityAlert, getLastKnownLocation, requestLocationUpdates
SMSManager	copyMessageToIcc, deleteMessageFromIcc, sendTextMessage
TelephonyManager	getCellLocation, getDeviceID
ContactsProvider2	delete, insert, query, readAccess, update, writeAccess
MMSSMSProvider	delete, insert, query, update
TelephonyProvider	delete, insert, query, update
SettingsProvider	delete, insert, query, update

而 ActivityManagerService 则负责管理 App 应用 Activities 的栈、Activity 生命周期，以及 Intent 的广播。它作为一个 USOM，还负责给 Activity 和 Intent 等资源打上基于 TE 的安全标记，以实现它们访问的安全加固。例如当执行设置某 Activity 到前台或后台的操作时，ActivityManagerService 中对应 ActivityStack，就会让 USSS 根据主体(App)类型和客体(该 App 的 activity)类型查询该操作是否被许可。同样，ActivityManagerService 也会根据安全属性，过滤发送者否可广播 Intent，或接收者是否可接收 Broadcast Intent。

在 Android 中除了 activity，还有个 ContentProvider 组件，该组件是 Android App 应用间共享数据的首先途径。表 1 中所列的 ContactProvider2、SettingsProvider 等，都是基于 ContentProvider 实现的 App 应用。这些应用作为一个 USOM，还要负责对其中的数据条目打上安全标签，以便在 USSS 保护下，实现这些数据安全的增、删、改、查等操作。由于 ContentProvider 是基于 SQL-Lite 数据库的实现，在该类 USOM 中的访问控制有两种粒度：一是以 SQL-Lite 的数据库为单位；另一则是单条数据。

Service 通过 ICC 向 Android 的其它组件提供功能服务。这些 services 也可作为 USOMs，将被调用的服务纳入 MMAC 保护。基于设置 service 的 Type 以及该服务下诸功能类型的安全属性，这些服务通过 AIDL 以 Binder IPC 对象的方式，在 TDSD-Droid 中实现基于 TE 的访问控制。

为了真正实现上面所述 MMAC，TDSD-Droid 扩展了基于 TE 的 SELinux 安全策略语言，引入了内核层外新的客体类别：Activity、Service、ContentProvider 和 Intent，以及与这些类别相关联的操作。这些类别被用来描述位于 Android 用户空间的资源客体。

4.1.3 FSP 实现

在 TDSD-Droid 中，安全环境切换器 SES(Security Environment Switcher)是 FSP 的核心，负责根据用户角色、网络、

时间及地点等条件变化而给 Android 终端运行配置不同的安全环境。基于不同的安全环境，相同主体将拥有客体不同的访问权限。SES 在 SystemServer 中驻留了几个 Listener 线程，以监控具体网络环境等参考环境变量的变化，修改 USSS 与 KSSS 的安全运行环境变量，最终改变了各主、客体的安全属性与安全策略，从而实现了 TDSD-Droid 随环境变化而适配不同安全策略的灵活安全策略。

为了支持 SES 的实现，TDSD-Droid 又扩展了安全策略语言，增加了 option 关键词，以支持与安全场景相关的安全策略的禁止与使能。同时，还引入了 switcherBoolean 声明，以将相应的安全场景映射成 booleans 值。而此 booleans 值将影响 option 指定下安全策略的禁止还是使能，进而映射现实 Android 运行的安全环境。参见下面一安全策略代码清单。从中看到，一旦布尔变量 allowIPTablesExec_b 被置为真，Android 终端就在 allowIPTablesExec_con 安全环境下运行。而标记有 option allowIPTablesExec _con 的安全策略将被使能。另外在清单中，还有诸如 developer_b 的布尔变量。这些变量可以面向用户空间，也可作用于内核空间。

```
bool developer_b = false;
kbool allowIPTablesExec_b = true;
… …
switchBoolean:allowIPTablesExec_b {
    context= allowIPTablesExec _con;
}
… …
option allowIPTablesExec _con {
… …
}
```

FSP 机制实现中除了实现 SES 组件和扩展安全策略语言外，还支持安全策略来源多样化，以及安全策略动态加载。除了通过预加载的策略库来标记随机器启动的各 Services 和 App 应用外，TDSD-Droid 还支持第三方应用在动态安装时，选择成为一个 USOM。此时该 App 提供自己安全策略，并标记第三方应用主体和属于该 App 的客体资源的安全属性。为此 TDSD-Droid 扩展了 PackageManagerService 在该类第三方应用安装时，标记新安装应用及其资源的安全属性，抽取该 App 专有策略文件注入到 USSS，存放到用户空间安全策略库。

为了完全支持上述第三方安全策略的动态注入，有个难题一定要解决，那就是新策略与已有策略发生冲突怎么办？比如，某主体对某客体的 query 操作，其自己的策略是 allow，而系统缺省策略是 neverallow，那应遵循那条策略设定？FSP 又引入了安全策略来源优先权重选择机制。该机制对可能的安全策略来源设置了一个 priority 值，该值越大优先权越大。当优越权更大的某条安全策略与优先权更低的相冲突时，FSP 将屏蔽优先权更低的安全策略。事实上，规避安全策略冲突的方法还有很多。TDSD-Droid 还实现了单调增强原则：遵循该原则，选择限制性最强的安全策略；这原则也有效解决了安全策略冲突难题，特别适合某些对安全要求特别高的运用环境。

4.2 安全策略学习机制

4.2.1 安全策略学习机制实现

TDSD-Droid 除了实现了内核层和上层基于 TE 的强制访问控制机制，还实现了安全策略学习机制。该机制实现落在 USSS 与 KSSS 扩展、reportService 以及运行于后台服务器的 studyService 上。

TDSD-Droid 会提供一份参考安全策略。USSS 或 KSSS 发现不同于参考安全策略的新策略，将其收集，并通过 reportService 上报给远程的 studyService。其中 reportService 被实现为一个 Android Service，驻留在 Android 的 SystemServer 中，而 studyService 则是条后台 Web Service，驻留在后台服务器。studyService 在接收到新的策略后，通过自动学习机制，判断并决定是否将该策略纳入参考安全策略库中。Study Service 的处理流程如图 2 所示。

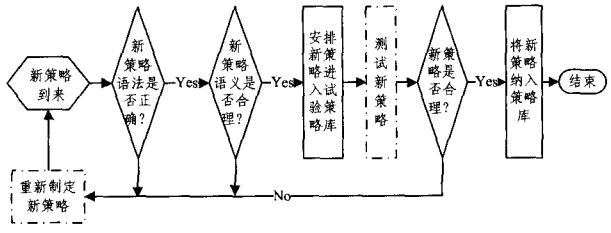

图 2 studyService 工作流程

图 2 还有个测试新策略步骤，需要相应测试团队和手动参与相应管理制度，所以用了虚框。基于该学习机制，安全策略将面对复杂的运用环境，得到不断的改善。尤其是当封装有 TDSD-Droid 的 Android 终端，配上该 studyService 运行在某行业领域时，该学习型机制将在保障这些终端安全的同时越来越贴近它们的实际运用。

TDSD-Droid 与后台交互的策略定义中，设计了一个 level 字段，以标识该安全策略是针对上层 MMAC 的，还是针对内核层 MAC 的。

4.2.2 远程安全策略下发机制实现

为了测试或运用通过该机制获得的新安全策略，TDSD-Droid 还实现了远程安全策略动态下发机制：RemotePolicyService 来接收后台下发的新策略。该服务驻留在 SystemServer 中，其工作流程如图 3 所示。

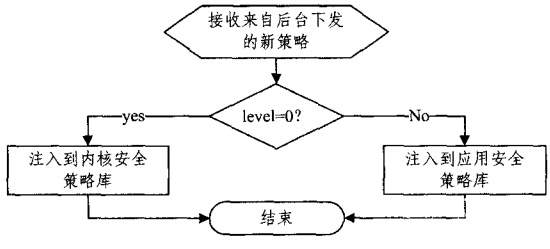

图 3 RemotePolicyService 工作流程

4.3 完整性验证实现

TDSD-Droid 作为一个立体防御体系，实现了以 TF 智能卡为可信根的完整性验证功能：在 TF 智能卡的片内操作系统 COS(ChipOperatingSystem)中实现了 SHA 摘要算法。基于 PKI 密钥和 SHA 算法，可顺利检查出核心模块是否被篡改。整个完整性验证的步骤如下：

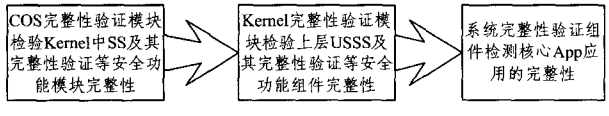

图 4 TDSD-Droid 完整性验证步骤

完整性验证核心算法在 TF 智能卡的 COS 中实现，并提供了相应的接口供内核完整性验证模块和 Android 中间层的完整性验证组件调用。这些接口定义参见如下代码清单：

```
// 初始化 TF 智能卡接口
unsigned long CALLBACK SD_Initialize();
// 退出 TF 智能卡接口
unsigned long CALLBACK SD_Finalize();
// 导入验算 PKI 公钥
unsigned long CALLBACK SD_ImportKey(
  DEV_HANDLE hDev,
  unsigned long ulKeyType,
  unsigned char * puchKey,
  unsigned long ulKeylen,
  unsigned long ulFlags,
  KEY_HANDLE * hKey
);
// 导出验算 PKI 公钥
unsigned long CALLBACK SD_ExportKey(
  DEV_HANDLE hDev,
  KEY_HANDLE hKey,
  PSDT_PUBKEY lpPublicKey,
  unsigned char * puchKey,
  unsigned long * ulKeylen,
  unsigned long ulFlags
);
// 删除验算 PKI 公钥
unsigned long CALLBACK SD_DelKey(
  DEV_HANDLE hDev,
  KEY_HANDLE hKey,
  unsigned long ulKeyType,
  unsigned long ulFlags
);
…… ……
// SHA 报文摘要计算
unsigned long CALLBACK SD_digest_compute(
  DEV_HANDLE hDev,
  unsigned long ulKeyType,
  unsigned long ulHashType,
  unsigned char * puchData,
  unsigned long ulDatalen,
  unsigned char * puchSign,
  unsigned long * pulSignlen,
  unsigned long ulFlags
);
// SHA 报文摘要比对
unsigned long CALLBACK SD_digest_compare(
  DEV_HANDLE hDev,
  unsigned long ulKeyType,
  unsigned long ulHashType,
  unsigned char * puchData,
  unsigned long ulDatalen,
  unsigned char * puchSign,
  unsigned long * pulSignlen,
  unsigned long ulFlags
);
```

TDSD-Droid 充分利用 SD 卡已是 Android 设备必备器件的特点，以及 SD 卡的可拆卸，利用自主定制的 TF 智能卡实现了灵活的完整性验证可信根。TF 智能卡的 COS 作为与 Android 并行运行且相互独立的操作系统，其在初始化过程中，验证内核中核心安全模块的完整性，进而判断这些模块是否被篡改。如果发现被篡改，直接在该初始化流程中启动关机流程，从而导致装有非法内核的 Android 不能正常开机工作。当用户不需要此安全特性时，可以将 TF 智能卡替换为普通 SD 卡，但此时就不能保证 Android 终端确实运行在 TDSD-Droid 保护下。

TDSD-Droid 在内核层实现了一个完整性验证模块：integrityverify.o；在用户空间实现了一个完整性验证服务：integrityVerifyService。该服务也驻留在 Android 的 SystemServer 中。TDSD-Droid 根据 Android 系统分成内核、系统框架和 App 应用等多层，且上层依赖于下层的特性，以 TF 智能卡为完整性验证的根，一环套一环验证：针对内核与系统框架层的不合法，将禁止加固后的机器运行，而针对 App 应用的不合法，只是禁止该 App 应用运行。

5 测试与验证

5.1 私密信息保护

Android 系统 services 和 ContentProviders 都是 Android 系统的关键部件。访问这些 Services 和 ContentProviders 的接口，标准 Android 的 ASF 会进行访问许可判决。问题是这些许可是针对 Service/ContentProvider 整体。一旦被许可，就可访问 Service/Provider 的所有接口。这种访问控制管理太粗，不能针对这些组件的功能或数据进行更精细的权限授予，也就不能对这些敏感资源划定谁可访问，什么时机可以访问，以及如何访问。像 Facebook、WhatsApp[15] 之类的应用，就能对整个 contacts 数据库做任何访问，但这些程序功能上只需要数据库中有限的数据：姓名、电话号码和 email 地址；结果在标准 Android 中，向这些程序提供了更多私密信息。通过标准 Android 用户所受到的攻击发现，正是这些过多权限的应用程序被不良程序插入后，导致私有数据（个人的家庭住址、信用卡号、甚至密码等）被盗，甚至经济受损。

而标准 Android 被 TDSD-Droid 加固后，针对每个 Service的接口和功能，都会有自己的安全策略，并接受 USSS 的访问许可判决，从而对这些敏感资源进行了更精细粒度的保护。经测试，经过 TDSD-Droid 改造过的 Android，利用 Facebook 等的不良程序已无法获得正常程序所需要数据之外的数据。

5.2 篡改攻击防护

现在比较难以防范的篡改攻击，是基于自身修改代码 SMC(Self-Modifying Code)技术的动态篡改[16]。这类程序能够使得运行的代码与程序执行前的静态二进制代码不相同，进而隐藏实际代码、数据及程序的执行流等，因而被许多恶意篡改软件使用。利用 SMC 的恶意篡改代码行为一般分为 3 步：一是冒充为目标进程，二是修改存放代码的内存页的读写权限，三则是针对原始代码进行覆盖篡改。

事实上，经过 TDSD-Droid 加固过的 Android 系统，由于拥有了更强安全属性划分、更精细化的访问控制管理，其实际

目标进程很难被冒充，代码内存的读写权限也很难非法获得。为了测试，了编写一个采用 SMC 技术的 App 应用程序，并故意放开了 setuid、domain_transition 和代码内存的读写权限；同时在 Linux 下编写了一个控制台程序，基于 SMC 技术，利用控制台键盘输入来覆盖该 Apk 应用程序在内存中的代码。最终测试发现，TDSD-Droid 中以页为单位的完整性验证组件，很容易检测到该 App 应用程序被篡改，并有效地中止该应程序的执行。

6 评估

这节从安全策略设计合理性、安全防护有效性和安全加固性能变化的角度来评估 TDSD-Droid，给出该安全加固系统可行性评估结果。

6.1 安全策略

众所周知，安全策略设计是个繁重的任务。但 Android 终端用户，使用的基本都是预装的应用和服务。为此 TDSD-Droid 设计了一套参考安全策略库，涵盖 MAC 和 MMAC，其中定义了 102 个 classes，383 个 types，以及数以千计的访问控制规则，很好地对内核和用户空间中的主体、客体进行了安全隔离，对主体对客体的访问实施了更精细化控制。

另外，TDSD-Droid 中的 FSP 机制，允许第三方应用自定义安全策略，对自身的资源或功能实施 MMAC 保护。同时，安全策略学习机制使得 TDSD-Droid 具有了学习改进能力，使得它可以向有实战经验的安全管理员和第三方应用开发人员学习，将合理的安全策略吸收进参考安全策略，从而使得安全策略不断被改善，贴近实际安全防护需求。

6.2 攻击防护

通过收集针对性 Android 攻击程序，进行加固后防护测试。测试结果如表 2 所列。

表 2 TDSD-Droid 攻防测试

攻击类型	测试程序	测试结果
利用 Root 权限攻击	mempodroid Exploit	由于内核 MAC 保护，有效地减轻了该类攻击的威胁。
通过 root 执行 App 攻击	Synthetic Test App	由于用户空间 MMAC 和内核 MAC 保护，极大地减弱了该类攻击的威胁。
利用权限过大 App 的信息偷窃攻击	Facebook WhatsApp	由于用户空间 MMAC，有效阻止了需求之外信息的被泄漏。
利用感应器的行为数据盗窃攻击	Synthetic Test App	由于定制了 SensorManager 等 USOMs 和用户空间 MMAC 基于安全环境敏感，有效地阻止了感应器相关数据外泄。
利用职责不清的攻击	Synthetic Test App	由于用户空间 MMAC，对 ICC 间交互的 Intents 实施了更强的限制，更精细地划分了各组件的职责，有效阻止了职责之外的响应。
内外勾结攻击	Synthetic Test App	由于完整性验证机制，从一定程度上也阻止了该类的攻击。

从表 2 可以看到 TDSD-Droid 针对各类攻击都进行了有效的防御，甚至对别的防御手段毫无办法的感应器行为推断和内外勾结之类攻击，也有不俗的表现。当然对于内、外全面的勾结，比如连 TF 智能卡可信根之类防护基本设施也破坏，将整个防御体系改得面目全非的攻击，TDSD-Droid 目前也是力有不逮。

事实上，TDSD-Droid 与其它现有的 Android 加固技术相比，也有全面更优的表现，如表 3 所列。

表 3 TDSD-Droid 等 Android 安全加固技术对比

加固技术	MAC	MMAC	RBAC	完整性验证	动态安全策略
TDSD-Droid	支持	支持	支持	支持	支持
SEAndroid	支持	支持(仅安装时有效)	不支持	不支持	仅支持本地动态更新
TrustDroid	支持	不支持	不支持	不支持	不支持
Saint	不支持	支持(偏重于 App 间隔离)	不支持	不支持	不支持
TaintDroid	不支持	支持(偏重于敏感数据保护)	不支持	不支持	不支持
XManDroid	不支持	支持(偏重于特权攻击侦测)	不支持	不支持	不支持

从表 3 可以看到，TDSD-Droid 是最完善的 Android 安全加固技术。Saint、TaintDroid 与 XManDroid 等基于用户空间的加固的技术，过于片面，要想真正发挥作用还有赖于内核 MAC 技术，以及自身所没有的用户空间安全加固技术。

6.3 总体性能

通过改造前、后相关执行速度、所占存储空间参数的比较，表 4、表 5 展示了 TDSD-Droid 加固对 Anroid 4.1.2 的机器 HD4 所带来的性能上的影响。

表 4 TDSD-Droid 改造前后 Image 大小与启动时间对比

Android 状态	Boot image (kB)	System image (kB)	Recovery image(kB)	开发启动时间(s)
改造前	4094	248600	4444	20.7
改造后	4238	250644	4608	22.3

从表 4 的数据来看，TDSD-Droid 改造后 Image 会有所变大，对于目前以 GB 为单位的 Android Flash 空间来讲，这点空间的增加是完全允许的。开机启动也显得稍慢了些，但 1～2s 延缓也在用户的许可范围之内。

表 5 TDSD-Droid 改造前后 AnTutu Benchmark[17] 对比测试

测试项目	标准 Android	TDSD-Droid
RAM 性能	1186	1119
CPU 整数性能	1364	1310
CPU 浮点性能	1121	1096
2D 绘图性能	358	[540×960]379
3D 绘图性能	1664	[540×960]1589
sd 写入速度	(5.0MB/s) 50	(5.4MB/s) 54

从表 5 的数据来看，TDSD-Droid 各项性能指标总体有所下降，但下降极为有限，一般用户根本体验不到 TDSD-Droid 所带来的性能差异。

6.4 尚待完善的工作

首先是 FSP 中 SES 组件尚待完善，主要是其中安全场景的划分与实际安全运用场景还不是十分相符。由于 Android 终端的用户众多，运用的场景也是千变万化，要为 SES 定义符合安全保护实战要求的角色与安全场景，进而为它们各自制定适配的安全策略，确实是件困难的事情。目前，TDSD-Droid 正为某行业定制安全加固 Android 终端，在此过程中，在为之开发一个合理的 SES 组件，并希望寻求一套方便的定制方法来将该组件拓展到其它行业或领域。

其次是安全策略学习机制尚待完善，主要是判决新安全策略合理性还不够自动化。虽然该学习机制在语法、语义等方面，对新策略好坏的判定，起到了一定的辅助作用，但由于用户的使用习惯不同，Android 终端上运行软件的差异性，以

(下转第 250 页)

dings of NAACL HLT, 2007. Rochester NY, 2007: 97-104

[6] Zha Hong-yuan. Generic Summarization and Key Phrase Extraction using Mutual Reinforcement Principle and Sentence Clustering[C]// Proceedings of ACM SIGIR, 2002. Tampere Finland, 2002: 113-120

[7] Wei Fu-ru, Li Wen-jie, Lu Qin, et al. Applying Two-Level Reinforcement Ranking in Query-Oriented Multidocument Summarization[J]. Journal of the American Society for Information Science and Technology, 2009, 60(10): 2119-2131

[8] Wei Fu-ru, Li Wen-jie, Lu Qin, et al. Query-Sensitive Mutual Reinforcement Chain and Its Application in Query-Oriented Multi-Document Summarization[C]// SIGIR, 2008. Singapore, 2008

[9] Bollegala D, Matsuo Y, Ishizuka M. Measuring Semantic Similarity between Words using Web Search Engines[C]// Proceedings of WWW. 2007: 757-766

[10] Cilibrasi R L, Vitanyi P M B. The Google Similairity Distance [J]. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(3): 370-383

[11] Sahami M, Heliman T D. A Web-based Kernel Function for Measuring the Similarity of Short Text Snippets[C]// Proceedings of WWW. 2006: 377-386

[12] Che Wan-xiang, Li Zheng-hua, Liu Ting. LTP: A Chinese Language Technology Platform[C]// Proceedings of the Coling 2010: Demonstrations. Beijing, China, 2010: 13-16

[13] Lin C-Y, Hovy E H. Automatic Evaluation of Summaries Using N-gram Co-occurrence Statistics [C]// Proceeding of 2003 Language Technology Conference (HLT-NAACL 2003). Canada, 2003

[14] Fellbaum C. WordNet [Z]. Theory and Application of Ontology: Computer Applications, 2010: 231-243

[15] Jones K S. A Statistical Interpretation of Term Specificity and Its Application in Retrieval[J]. Journal of documentation, 1972, 28(1): 11-21

(上接第 234 页)

及安全策略管理员制定策略视角的不同，都会导致安全策略合理性被质疑。因此在 TDSD-Droid 学习机制中，引入了人为的处理流程，特别强调有经验安全策略管理员的参与。在将来，希望能为该安全策略学习机制开发出一套仿人工智能测试系统，以便实现安全策略学习更全面的自动化。

结束语 这篇文章展现了 TDSD-Droid 基于强制访问控制机制的立体安全防御系统设计与实现，介绍了 TDSD-Droid 的架构，上、下一致的 MMAC 与 MAC 强制访问控制机制、FSP 灵活性安全策略机制，以及完整性验证机制等。本文也演示了 TDSD-Droid 面对 Android 攻击的防御功效，并给出了相关的性能测试结果。

参 考 文 献

[1] Llamas R, Restivo K, Shirer M. Android Marks Fourth Anniversary Since Launch with 75.0% Market Share in Third Quarter [EB/OL]. https://www.idc.com/getdoc.jsp?containerId=prUS23771812, IDC, 2012

[2] Kleidermacher D, Kleidermacher M. Embedded System Security Practical Methods for Safe And Secure Software and Systems Development [M]. Waltham, MA, USA: Elsevier Inc, 2012: 4-24

[3] Armando A, Merlo A, Verderame L, et al. An Empirical Evaluation of the Android Security Framework [C]// Proceedings of the 28th IFIP TC-11 International Information Security and Privacy Conference (SEC 2013). Auckland: Springer, 2013: 176-189

[4] Smalley S, Craig R. Security Enhanced (SE) Android: Bringing Flexible MAC to Android [C/OL]. http://selinuxproject.Org/~se-android/papers/NDSS2013-SEAndroid-Paper.pdf, NDSS, 2013

[5] Jhswx84. SELinux 详解[M/OL]. http://wenku.baidu.com/view/4d26594fc850ad02de804189.html, Baidu, 2012

[6] Enck W, Ongtang M, McDaniel P. Understanding Android security [J]. IEEE Security and Privacy Magazine, 2009 7(1): 50-57

[7] Sally. SELinux 学习笔记[M/OL]. http://wenku.it168.com/d_001220063.shtml. IT168, 2013

[8] Spencer R, Smalley S, Loscocco P, et al. The Flask security architecture: System support for diverse security policies [C]// Proceedings of The Eighth USENIX Security Symposium. Washington: USENIX, 1999: 123-139

[9] Carter J. Using gconf as an example of how to create an userspace object manager [C/OL]. http://www.nsa.gov/research/_files/selinux/papers/gconf07-paper.shtml, NSA, 2009

[10] NSA. SE For Android [EB/OL]. http://selinuxproject.org/page/SEforAndroid. NSA, 2013

[11] Ongtang M, McLaughlin M, Enck W, et al. Semantically rich application-centric security in Android[J]. Security and Communication Networks, 2012, 5(6): 658-673

[12] Bugiel S, Davi L, Dmitrienko A, et al. Practical and Lightweight Domain Isolation on Android[C]// Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM '11). New York: CCS, 2011: 51-62

[13] Enck I, Gilbert P, Chun B, et al. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones[C]// proceeding of: 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010. Vancouver, BC, Canada: USENIX, 2010: 1-6

[14] Bugiel S, Davi L, Dmitrienko A, et al. Towards Taming Privilege-Escalation Attacks on Android [C/OL]. http://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/NDSS_2012_Towards_Taming_Privilege-Escalation_Attacks_on_Android.pdf, NDSS, 2012

[15] Bea F. WhatsApp reads your phone contacts and is breaking privacy laws [CP/OL]. http://www.digitaltrends.com/mobile/whatsapp-breaks-privacy-laws/, DTDigital Trends, 2013

[16] Cai H, Shao Z, Vaynberg A. Certified Self-Modifying Code [C]// Proceedings of 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation. San Diego: PLDI' 2007: 66-77

[17] AnTuTu Labs. AnTuTu Benchmark [CP/OL]. https://play.google.com/store/apps/details?id=com.antutu.ABenchMark, Google, 2013