**毕业论文（设计）外文翻译**

Semantically richapplication-centric securityinAndroid

作者： Machigar Ongtang, Stephen McLaughlin, William Enck and Patrick McDaniel

国籍：

出处：IEEE 2009 Annual Computer Security Applications Conference

摘要

智能手机现在无处不在。实际上，这些相对较新的系统和应用程序仍然使用以前默认的安全要求。因此，当前智能手机操作系统的安全监测系统是很落后的。在本文中，我们考虑了智能手机应用程序和现有的Android操作系统与框架结合起来。我们提供安全的应用程序交互（缩写为Saint）是一种修改后的基础结构，它控制安装时权限分配及其运行时使用，这由应用程序提供程序策略决定。对应用程序策略的语义进行了深入的描述。确定Saint的体系结构和技术细节，并对其扩展、优化和改进方面进行了探讨。我们通过一个具体的例子和对真实应用程序的研究，Saint为应用程序在平台上断言和控制安全决策提供了必要的实用程序。

1. 引言

智能手机推动了移动计算的复兴。智能手机上运行的应用程序支持通信、娱乐和商业领域的广阔新市场。支持此类应用程序的硬件、访问和软件现在已广泛提供，而且通常价格便宜，例如苹果的应用商店、安卓的应用市场和黑莓的应用世界。因此，智能手机系统变得非常普及。

手机应用程序正从独立设计转向协作（服务）模式。在这个新兴的环境中，应用程序公开特定的内部特性给其他程序并且使用其他程序提供的服务。在后一种情况下，应用程序只是在运行时搜索和使用服务类型的适当提供者，而不是在开发期间自己实现特定的服务。这使得“使用和扩展”开发的丰富文化得以发展，从而导致了创新应用的爆炸式增长。这种文化很可能在Android操作系统社区中得到了最好的说明，也可以在其他智能手机操作系统（如Access Linux平台）中使用。此外，iOS（以前称为iPhone OS）也朝着这个应用程序模式迈出了一步。特别是，最近发布的iOS4支持多任务处理，这使得iPhone可以同时运行多个第三方应用程序。

安卓系统（以及许多其他手机操作系统）的安全模式是“以系统为中心”。应用程序静态地标识在安装时控制其数据和接口权限的权限。但是，应用程序/开发人员此后对这些权利授予的对象或后来如何行使这些权利的能力有限。从本质上讲，权限被断言为对应用程序所需的保护类型的模糊建议。应用程序必须相信，操作系统和用户会做出正确的选择，决定哪些应用程序可以授予这些权限，在许多情况下，这是不可能的，因为它们没有足够的目的来实现这一点。

假设一个PayPal的服务构建在Android上。诸如浏览器、电子邮件客户端、软件市场、音乐播放器等应用程序使用PayPal服务来购买商品。在这种情况下，PayPal服务是一个应用程序，它声明必须授予使用其接口的其他应用程序的权限。什么是合法申请？只有PayPal应用程序（PayPal公司）才知道这个问题的答案。这不仅仅是一个简单的问题，即哪个应用程序正在发出请求（在许多情况下，在Android中，这对于被调用的应用程序来说是不可知的），还包括请求发出的位置、时间、方式和条件。不幸的是，Android没有提供任何方法来回答这些问题或根据这些问题执行安全策略。简单地说，Android系统保护手机免受恶意应用程序的攻击，但为应用程序提供的保护自身的基础设施非常有限。基于Android应用程序的广泛开发，我们观察到Android安全框架中应用程序不可用的三个基本应用程序策略。

1. 权限分配策略——应用程序控制授予谁访问其有限接口的权限的能力，例如白名单或黑名单应用程序。
2. Android的接口暴露策略——只为应用程序提供了一些基本的工具，以控制其他应用程序如何使用它们的接口。
3. 接口使用策略——应用程序在运行时选择所使用的应用程序接口的方法有限。

本文介绍了安全应用程序交互（Saint）框架，该框架通过解决这些关键应用程序需求的策略扩展了现有的Android安全体系结构。

在Saint增强的基础结构中，应用程序提供安装时策略，用于管理保护其接口的权限分配。在运行时，应用程序之间的访问或通信受调用方和被调用方应用程序断言的安全策略的约束。Saint策略通过基于运行时状态（例如位置、时间、电话或网络配置等）限制访问，远远超出了目前在Android中可用的静态权限检查。我们定义了Saint框架，并讨论了使用扩展的策略执行功能增强Android的复杂性，以及开发检测应用程序之间不兼容和依赖性的机制。

我们以一个激励性的例子开始讨论。

1. 智能手机应用程序安全

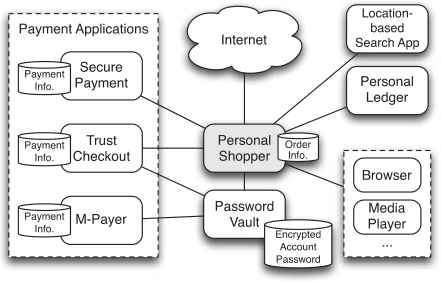


图 1

图1展示了虚构的PersonalShopper智能手机购物应用程序。 PersonalShopper跟踪用户希望购买的商品，并与支付应用程序进行交互以购买它们。用户通过电话的用户界面输入所需的项目（可能通过点击浏览器，媒体播放器等上的项目），创建独立于“购物车”的供应商。用户随后以两种方式之一获取物品。 用户可以通过单击来指示应用程序“查找”项目。 在这种情况下，应用程序将搜索已知的在线供应商或购物搜索站点（例如，Google产品搜索）以找到所需的项目。在多个供应商提供相同项目的情况下，用户通过菜单选择他们的供应商选择。 用于查找产品的第二种装置是通过地理位置——例如，通过基于位置的搜索应用程序，可以向移动通过商场的用户警告物理商店中可用的项目的存在。 在这种情况下，她将被引导到实体供应商以获得该项目。无论物品如何被发现，PersonalShopper的第二个目标是促进购买过程本身。 在这种情况下，它适用于我们的示例结帐应用程序SecurePayer和TrustCheckout。 PersonalShopper访问结账应用程序，并充当买方和商家之间的中间人，以提高购物效率和保护客户隐私。 他们使用的应用程序和服务将与密码保险库进行交互，以提供身份验证凭据。 完成后，交易记录在个人分类帐申请中。

考虑一下这个应用程序建议的一些（很多）安全要求：

1. PersonalShopper应该只使用可信赖的支付服务。 在图1中，它可以信任SecurePayer和TrustCheckout，但不信任其他未知的支付提供商（例如，M-Payer提供商）。
2. PersonalShopper可能只想在安全的条件下将服务的使用限制为仅受信任的网络。 例如，它可能希望在电话使用不受保护的WiFi连接时禁用搜索。
3. PersonalShopper可能需要使用某些版本的服务软件。 例如，密码保险库应用程序V.1.1可能包含泄漏密码信息的错误。 因此，应用程序将要求密码库为V.1.2或更高版本。
4. PersonalShopper可能希望确保交易信息不会被手机的分类帐应用程序泄露。因此，应用程序希望仅使用无法访问Internet的分类帐。
5. 向PersonalShopper提供功能的应用程序也可能对其提出安全要求。 例如，为了保护位置隐私，基于位置的搜索应用程序可以仅在PersonalShopper保持访问位置信息本身的权限的情况下仅提供PersonalShopper位置信息，例如，电话的GPS服务。

当前的Android安全系统不支持这些策略。 虽然其中一些可能使用复杂的应用程序代码，代码签名和权限结构的组合进行部分模拟，但它们只是在Android安全策略的范围之外。 因此（以及我们在Android中构建系统的丰富经验的核心），应用程序必须在Android系统当前提供的基本结构之上拼凑自定义安全功能。 在可能的情况下，这个过程是临时的，容易出错的，重复的和不精确的。

所需要的是Android为应用程序提供更加语义丰富的策略基础结构。 我们通过概述Android系统和安全机制开始我们的调查。 第4节研究了满足应用程序安全要求可能需要的一系列策略，突出了当前Android无法满足的策略。 然后，我们介绍了Saint系统的目标，设计和实现。

1. Android

Android是由Google领导的开放手机联盟开发的手机平台。

该平台因其开源性质以及全球电信提供商的采用而迅速在开发者社区中受到欢迎。

虽然Android基于Linux，但提供给应用程序开发人员的中间件隐藏了传统的操作系统抽象。 该平台本身专注于应用程序，并且许多核心手机功能以与第三方开发人员使用的相同方式实现为应用程序。

Android应用程序主要用Java编写，并编译成自定义字节码（DEX）。 每个应用程序都在一个独立的Dalvik虚拟机解释器实例中执行，该实例作为唯一的用户身份运行。从底层Linux系统的角度来看，应用程序表面上是孤立的。此设计最大限度地降低了折衷的影响，例如，被利用的缓冲区溢出仅限于应用程序及其数据。

所有的应用程序间通信都通过中间件的绑定进程间通信（IPC）机制（我们的讨论假设所有IPC都是绑定器IPC）。Binder为应用程序执行提供基本功能。 应用程序由组件组成。 组件主要使用Intent消息进行交互。 Intent包括两条主要信息，即接收组件的地址和要传递给这种组件的数据。尽管Intent消息可以通过名称显式地寻址应用程序中的组件，但更常见的方式是使用操作字符串隐式地解决Intent，该操作字符串描述已发生的操作或要执行的操作。操作字符串可以是Android的预定义操作之一，也可以是开发人员定义的操作字符串。 收件人组件通过定义指定一个或多个操作字符串的Intent过滤器来声明他们希望接收Intent消息。 当Intents被发送出去时，Android中间件会根据声明的断言自动解析相应的组件以处理此类事件。

有四种类型的组件用于构建应用程序; 每种类型都有特定的目的。 Activity（活动）组件通过触摸屏和键盘与用户连接。 通常，应用程序中的每个显示屏幕都是不同的活动。 一次只有一个活动处于活动状态，并且无论应用程序如何，所有其他活动的处理都将暂停。Service（服务）组件提供后台处理，以便在应用程序的活动保持关注时使用。 服务还可以导出远程过程调用接口，包括对回调的支持。 Broadcast Receiver（广播接收器）组件为异步事件通知提供通用机制。传统上，Broadcast Receiver（广播接收器）接收用动作字符串隐式寻址的Intent。 标准操作字符串包括“已完成引导”和“已接收信息”的事件。最后，Content Provider（内容提供者）组件是在应用程序之间共享数据的首选方法。 内容提供者应用程序编程接口（API）实现类似结构化查询语言（SQL）的接口; 但是，后端实现由应用程序开发人员完成。 API包括对读取和写入数据流的支持，例如，如果Content Provider（内容提供者）共享文件。 与其他组件类型不同，内容提供程序不通过Intent进行寻址，而是通过内容统一资源标识符（URI）进行寻址。 它是我们关注的应用程序组件之间的交互。 图2描绘了组件类型之间的通用IPC（进程间通信）。

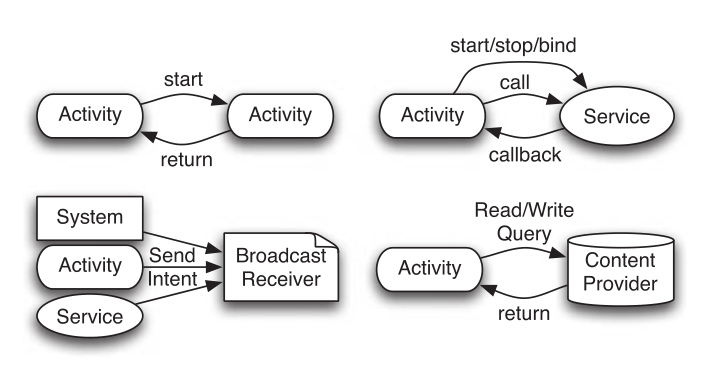


图 2

Android的应用程序级安全框架基于中间件参考监视器中强制执行的权限标签。权限标签只是一个唯一的文本字符串，可由OS和第三方开发人员定义。 Android定义了许多基本权限标签。 从以操作系统为中心的角度来看，应用程序是静态分配的权限标签，指示在运行时可访问的敏感接口和资源; 安装后权限集不能增长。应用程序开发人员在其包清单中指定应用程序所需的权限标签列表; 但是，并不总是授予请求的权限。

权限标签定义分布在框架和包清单文件中。 每个定义都指定“保护级别”。保护级别可以是“正常”，“危险”，“签名”或“签名或系统”。在应用程序安装时，将查询所请求权限的保护级别。始终授予保护级别为正常的权限。 如果安装了应用程序，则始终授予具有危险防护等级的权限; 但是，用户必须一起确认所有请求的危险权限。 最后，签名保护级别在没有用户输入的情况下影响权限授予。每个应用程序包都由开发人员密钥签署（与包含操作系统定义的权限标签的框架包一样）。 只有在请求签名保护的应用程序由签署定义权限标签的程序包的相同开发人员密钥签名时，才会授予签名保护权限。许多操作系统定义的权限使用签名保护级别来确保仅授予操作系统供应商分发的应用程序访问权限。 最后，“签名或系统”保护级别与签名级别的操作相同，但此外，权限授予由用于系统映像的密钥签名的应用程序。

权限标签策略模型还用于保护应用程序彼此之间的关系。 大多数权限标签安全策略在应用程序的包清单中定义。 如上所述，包清单指定了与应用程序功能要求相对应的权限标签。对应于应用程序的功能要求。 包清单还指定了用于保护每个应用程序组件的权限标签（例如活动，服务等）。简而言之，如果已经为应用程序分配了指定用于限制对目标组件的访问的权限标签，则应用程序可以使用另一个（或相同）应用程序中的组件启动IPC。 使用此策略和权限保护级别，应用程序开发人员可以指定其他应用程序如何访问其组件 有关Android应用程序级别安全策略及其细微之处的更完整描述，请参阅Enck等人的研究。

基于许可标签的安全策略源于手机开发的本质。 在许多方面，手动管理数百（数千）个潜在未知应用程序的访问控制策略是不可行的。 因此，Android通过让开发人员定义访问其接口的权限标签来简化访问控制策略规范。 开发人员无需了解所有现有（和未来）应用程序。 相反，权限标签允许开发人员间接影响安全决策。 但是，采用这种基于标签的策略导致了第1部分中介绍的Android安全框架的一些限制。

1. 申请政策

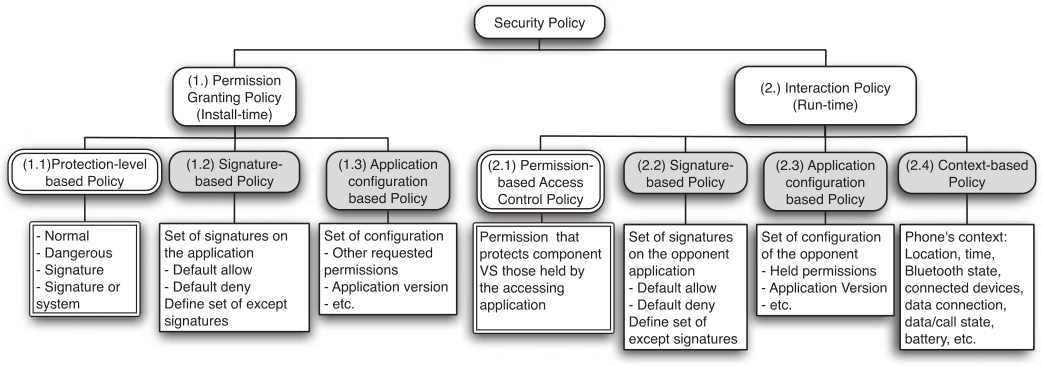


图 3

我们探索了无数的应用程序，作为理解适当的政策可表达性的一种手段。 初始策略分类如图3所示.Android仅支持图中所示的策略类的子集，如双笔划框所示。

许可授予策略（1）规定了权限分配。 除了使用Android的基于保护级别的策略（1.1）控制权限授予之外，应用程序A可能还需要基于签名的策略（1.2）来控制如何根据请求应用程序B（A和A）的签名授予其声明的权限。 B可以由不同的开发者密钥签名）。 相反，策略默认授予（或拒绝）权限，并使用例外列表拒绝（授予）由列出的密钥签名的应用程序。 应用程序还可能需要基于配置的策略（1.3）来基于请求应用程序的配置参数来控制权限分配，例如，请求的权限集和应用程序版本。

交互策略（2）规定应用程序与其对手之间的运行时交互。 应用程序A的对手是访问A资源的应用程序B，或者是A的动作目标，具体取决于访问控制规则（即，B是A的定义的对手，A是由A定义的规则的B的反对者）B）。 Android现有的基于权限的访问控制策略（2.1）提供了直接的静态策略保护，如第3节所述。但是，这种政策在很多情况下都是粗糙的，不够的。 应用程序可能需要基于签名的策略（2.2）来根据其签名限制对手应用程序的集合。 与上面提到的类似，需要default-allow（默认允许）和default-deny（默认禁止）模式。 使用基于配置的策略（2.3），应用程序可以定义对手应用程序的期望配置，例如，允许（或不允许）对手的最小版本和一组权限。最后，应用程序可能希望基于手机的瞬态来调节交互。 基于电话上下文的策略（2.4）基于诸如位置，时间，蓝牙连接和连接的设备，呼叫状态，数据状态，数据连接网络和电池电平之类的上下文来管理运行时交互。 请注意，最初，策略类型2.2和2.3可能看起来与1.2和1.3相同; 但是，前者类型也对目标应用程序提出了要求，无法用1.2和1.3表示。 但是，1.2和1.3是可取的，因为在适用时，它们没有运行时开销。

我们现在提供两个与我们的激励示例相关的示例应用程序策略，PersonalShopper，它与结帐应用程序，密码保险库，基于位置的搜索应用程序和个人分类帐交互。

安装时策略示例：在我们的PersonalShopper示例中，基于位置的搜索应用程序（com.abc.lbs）希望防止未经授权从其“QueryByLocation”服务泄漏位置信息。 当PersonalShopper请求用于保护“QueryByLocation”的权限com.abc.perm.getloc时，可以应用权限授予策略。 它需要基于应用程序配置的策略来指定对于要授予的权限com.abc.perm.getloc，请求者还必须具有“ACCESS\_LOCATION”权限。

运行时策略示例：为确保用于付款的结帐应用程序受信任，必须检查其签名。 PersonalShopper需要基于签名的策略来指定当源“Personal Shopper”（com.ok.shopper）使用操作“ACTION\_PAY”启动活动时，该策略确保已解析的应用程序由给定集合中的键签名。

1. 相关工作

最近手机安全的大部分工作都集中在应用程序安装时验证权限分配。 例如，Kirin强制执行安装策略，以验证应用程序请求的权限是否与系统策略一致。Kirin不考虑运行时策略，仅限于简单的权限分配。相反，开放移动终端平台提供的应用程序安全框架建议使用基于证书的机制来根据其来源确定应用程序的访问权限。 塞班在塞班签名计划中提供更严格的方案。在这个程序中，塞班基本上保证了应用程序的安全，并防止未签名的应用程序访问“受保护”的接口。 MIDP 2.0安全模型基于由移动信息设备配置文件（MIDP）实现者（例如，制造商和网络提供商）定义的保护域来调节敏感许可，例如网络访问或文件系统访问。

运行时策略系统欠发达。 Linux安全模块框架经常用于保护Linux手机。 例如，可信移动电话参考架构使用为移动电话平台开发的隔离技术实现了可信移动电话规范。Muthukumaran等人。 将SELinux安全策略应用于Openmoko，以确保手机和可信应用程序的完整性。 在相关工作中，Rao和Jaeger为智能手机开发了一个MAC系统，该系统使用来自多个利益相关者的输入来动态创建策略运行时权限分配。 Windows手机使用.NET紧凑框架，按合同的安全性，将每个应用程序绑定到在运行时强制执行的行为配置文件。该技术被进一步探索为安全执行潜在恶意代码的手段。 Windows手机也探索了系统调用插入等技术。 这些系统都不允许应用程序在其使用的接口和使用其接口的接口上放置上下文相关的策略。

1. 结论

在本文中，我们介绍了Saint框架。 Saint通过安装时权限授予策略和运行时应用程序通信策略解决了Android安全性的当前限制。 我们提供了操作策略，以揭示安全策略对应用程序功能的影响，并管理接口之间的依赖关系。在对许多应用程序的分析的推动下，我们的调查提供了相关安全上下文的初始分类。 最后，我们通过OpenIntents项目的案例研究证明了Saint的有用性，并表明Saint只会产生很小的性能开销，因此对于实际应用来说是实用的。

推动实际采用Saint的最迫切需求是将更多应用程序及其所需的策略集成到系统中。 我们寻求扩展圣政策以保护电话“系统”服务和蜂窝网络，并将其接口与广泛使用的安全基础设施（例如公钥基础设施和企业系统）集成。 通过持续的功能增强和用户研究，我们希望将Saint从一个研究系统转变为一个可行的框架，用于即将运行Android的数百万部手机。