**科技文献翻译**

原文题目：Android Permissions Demystified

译文题目： 神秘的Android权限

指导教师： 李晓宇 职称： 副教授

指导教师(校外)： 张创伟 职称： 研发经理

学生姓名： 李伟 学号： 20162430211

专 业： 软件工程

院（系）： 信息工程学院

完成时间： 2020年6月5日

2020 年 6 月 5 日

作者：Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, David Wagner

摘要

Android为第三方应用程序提供了广泛的功能包括访问手机硬件，设置和用户数据。访问与隐私和安全相关的部分API由安装时应用程序许可系统控制。我们研究Android应用程序，以确定Android开发人员是否遵循最低特权权限请求。我们构建了Stowaway，该工具可以检测已编译的Android应用程序中的特权。偷渡者确定应用程序使用的API调用集，并然后将这些API调用映射到权限。我们使用Android API上的自动化测试工具来构建检测超权限所必需的权限图。我们将Stowaway应用于940个应用程序，并且发现大约三分之一的人享有特权。我们调查特权过高的原因，并找到证据表明开发商正在尝试遵循最小特权，但有时由于API文档不足。

类别和主题描述符

D.2.5 [软件工程]：测试和调试；

D.4.6 [操作系统]：安全性和保护

一般条款

安全

关键字

Android, permissions, least privilege

# 介绍

Android不受限制的应用程序市场和开源使其成为第三方应用程序的流行平台。截至2011年，Android Market包含更多应用程序比Apple App Store [10]。 Android通过广泛的API支持第三方开发，该API为应用程序提供对电话硬件（例如摄像头），WiFi和蜂窝网络，用户数据和电话设置。

访问Android的与隐私和安全性相关的部分丰富的API由安装时应用程序权限系统控制。每个应用程序必须预先声明所需的权限，并且在安装过程中会向用户通知其将获得的权限。如果用户不想授予应用程序许可，他或她可以取消安装过程。

安装时权限可以为用户提供控制保护他们的隐私，并减少错误和漏洞对应用程序的影响。但是，如果开发人员通常要求安装时间许可系统无效超出所需的权限。特权过高的应用程序使用户面临不必要的权限警告和增加错误或漏洞的影响。 我们研究Android应用程序，以确定Android开发人员是否遵循最低特权或过度特权其应用程序。

我们提供了一种Stowaway工具，可以检测到特权过高在已编译的Android应用程序中。偷渡者组成分为两个部分：确定什么内容的静态分析工具API会调用应用程序生成的内容，以及一个标识每个API调用需要哪些权限。 Android的文档没有提供足够的权限信息以进行此类分析，因此我们根据经验确定了Android 2.2的访问控制策略。使用自动化测试技术，我们实现了85％的Android覆盖率API。我们的权限图可让您深入了解Android许可系统，使我们能够识别超权限。

我们将Stowaway从以下版本应用到940个Android应用程序：Android Market，发现大约三分之一的应用程序拥有特权。特权过高的应用通常要求的额外特权很少：仅一半以上包含一个额外的权限，只有6％的请求超过四个不必要的权限。 我们调查原因特权过剩，并发现许多开发人员错误源于对许可系统的困惑。我们的结果表示开发人员正在尝试遵循最小特权，支持诸如Android的安装时许可系统的潜在有效性。

Android提供了开发人员文档，但其权限信息有限。缺少可靠的权限信息可能会导致开发人员错误。该文档仅列出了78种方法的权限要求，而我们的测试则揭示了针对以下方法的权限要求1 259种方法（比文档多16倍）。此外，我们确定了Android中的6个错误权限文档。 这种不精确性使开发人员可以在猜测和留言板上添加参考资料。开发人员的困惑可能导致特权过高应用程序，因为开发人员添加了不必要的权限试图使应用程序正常工作。

贡献。我们作出以下贡献：

1. 我们开发了Stowaway，这是一种用于检测Android应用程序中超权限的工具。我们使用Stowaway和发现大约三分之一的人享有特权。
2. 我们确定并量化了导致特权过高的开发人员的错误模式
3. 我们使用自动化测试技术来确定Android的访问控制策略。与文档相比，我们的结果提高了15倍。

其他现有工具[11，12]和将来的程序分析可以利用我们的权限图来研究Android应用程序中的权限使用情况。可以在android-permissions.org上获得Stowaway和权限图数据。

组织。第2节概述了Android及其权限系统，第3节讨论了我们的API测试方法，第4节介绍了我们对Android API的分析。第5节介绍了用于检测过度特权的静态分析工具，而第6部分则讨论了我们的应用程序过度特权分析。

# ANDROID 权限系统

Android具有广泛的API和权限系统。我们首先提供Android应用程序平台和权限的高级概述。然后，我们将详细介绍如何强制执行Android权限。

## 2.1 Android 背景

Android智能手机用户可以通过 Android Market 或 Amazon Appstore 安装第三方应用程序。这些第三方应用程序的质量和可信度差异很大，因此Android将所有应用程序视为潜在的漏洞或恶意软件。每个应用程序在具有低特权用户ID的进程中运行，并且默认情况下，应用程序只能访问自己的文件。应用程序用Java编写（可能随附本机代码），并且每个应用程序都在自己的虚拟机中运行。

Android通过安装时权限控制对系统资源的访问。 Android 2.2定义了134个权限，分为三个威胁级别：

普通权限可以保护对API调用的访问，这些访问可能会惹恼但不会伤害用户。例如，SET\_WALLPAPER控制更改用户背景墙纸的功能。

危险权限控制对可能有害的API调用的访问，例如与花钱或收集私人信息有关的调用。例如，发送短信或阅读联系人列表需要危险权限。

签名/系统权限可控制对最危险特权的访问，例如控制备份过程或删除应用程序包的能力。 这些权限很难获得：签名权限仅授予使用设备制造商证书签名的应用程序，而SignatureOrSystem权限则授予使用特殊系统文件夹签名或安装的应用程序。这些限制实际上将签名/系统权限限制为预安装的应用程序，其他应用程序对签名/系统权限的请求将被忽略。

应用程序可以出于自我保护的目的定义自己的权限，但是我们专注于保护系统资源的Android定义的权限。在分析的任何阶段，我们都不会考虑开发人员定义的权限。同样，我们不会考虑包含在Google应用程序（例如Google Reader）中但不是操作系统一部分的Google定义的权限。

与系统API，数据库和消息传递系统进行交互时，可能需要权限。公用 API 描述8648个方法，其中一些受权限保护。用户数据存储在内容提供程序中，并且在某些系统内容提供程序上进行操作需要权限。例如，应用程序必须拥有READ\_CONTACTS权限才能在Contacts Content Provider上执行READ查询。 应用程序可能还需要获得许可才能从操作系统接收Intent（即消息）。Intent会通知应用程序事件，例如网络连接的更改，并且系统发送的某些Intent仅传递给具有适当权限的应用程序。此外，发送模拟系统Intent内容的Intent需要权限。

## 2.2 执行权限

我们描述了如何实现和保护系统API，内容提供者和意图。 据我们所知，我们是第一个详细描述Android权限执行机制的人。

### 2.2.1 API

API结构。Android API框架由两部分组成：一个驻留在每个应用程序的虚拟机中的库，以及一个在系统进程中运行的API的实现。API库以与其附带的应用程序相同的权限运行，而系统进程中的API实现不受限制。该库提供了与API实现进行交互的语法糖。库将在系统进程中将读取或更改全局电话状态的API调用代理到API实现。

API调用分为三个步骤（图1）。 首先，应用程序调用库中的公共API。其次，该库调用一个私有接口，也在该库中。专用接口是RPC存根。 第三，RPC存根通过系统进程启动RPC请求，该请求要求系统服务执行所需的操作。例如，如果应用程序调用ClipboardManager.getText（），则该调用将中继到IClipboard $ Stub $ Proxy，该代理将对系统进程的ClipboardService的调用代理。

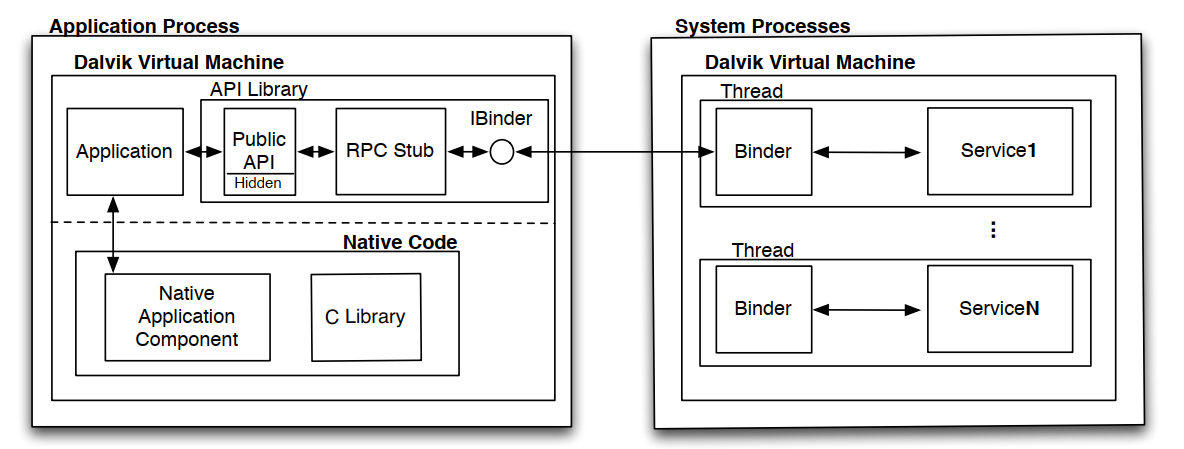
应用程序可以使用Java Reflection访问所有API库的隐藏和私有类，方法和字段。某些私有接口没有任何相应的公共API。但是，应用程序仍然可以使用反射来调用它们。这些非公共库方法仅供Google应用程序或框架本身使用，建议开发人员不要使用它们，因为它们可能在发行版之间更改或消失。但是，某些应用程序仍在使用它们。在系统进程中运行的Java代码位于单独的虚拟机中，因此不受反射的影响。

图1 Android 平台的架构。权限检查在系统进程中进行。

权限。为了实施权限，系统的各个部分调用权限验证机制来检查给定的应用程序是否具有指定的权限。权限验证机制是作为受信任系统过程的一部分实现的，权限验证机制的调用分布在整个API中。调用API时，没有用于检查权限的集中策略。而是，调解取决于权限验证调用的正确放置。

权限检查放在系统过程中的API实现中。必要时，API实现将调用权限验证机制以检查调用的应用程序是否具有必需的权限。在某些情况下，API库也可以冗余地检查这些权限，但不能依靠这些检查：应用程序可以通过RPC存根与系统进程直接通信来规避它们。因此，权限检查不应在API库中进行。相反，系统进程中的API实现应调用权限验证机制。

少数权限是由Unix组强制执行的，而不是Android权限验证机制。尤其是，当安装具有INTERNET，WRITE\_EXTERNAL\_STORAGE或BLUETOOTH权限的应用程序时，会将其分配给可以访问相关套接字和文件的Linux组。 因此，Linux内核针对这些权限实施访问控制策略。API库（具有与应用程序相同的权限运行）因此可以直接在这些套接字和文件上运行，而无需在系统进程中调用API实现。

本地代码。应用程序除了Java代码外，还可以包括本机代码，但是本机代码仍然属于许可系统。尝试打开套接字或文件是由Linux权限引起的。本机代码无法直接与系统API通信，相反，应用程序必须创建Java包装器方法才能代表本机代码调用API。执行API调用时，将照常实施Android权限。权限将应用于内容提供商存储的所有资源。通过将权限与路径（例如，content：// a / b）相关联，也可以以更精细的粒度应用限制。例如，既存储公共注释又存储私有注释的内容提供者可能想要为整个内容提供者设置默认权限要求，但随后允许无限制地访问公共注释。可以类似地为某些路径设置额外的权限要求，仅当调用应用程序具有提供者的默认权限以及特定于路径的权限时，才可以访问这些路径下的数据。

### 2.2.2 Content Providers

系统内容提供程序是作为独立的应用程序安装的，与系统进程和API库分开安装。它们使用静态和动态权限检查进行保护，并使用与应用程序可用的相同机制来保护自己的内容提供程序。

静态声明为给定的内容提供程序分配了单独的读取和写入权限。默认情况下，这些权限将应用于内容提供者存储的所有资源。通过将权限与路径（例如，content：// a / b）相关联，也可以以更精细的粒度应用限制。例如，既存储公共注释又存储私有注释的内容提供者可能想要为整个内容提供者设置默认权限要求，但随后允许无限制地访问公共注释。可以类似地为某些路径设置额外的权限要求，仅当调用应用程序具有提供者的默认权限以及特定于路径的权限时，才可以访问这些路径下的数据。

### 2.2.3 Intents

Android的Intent系统已广泛用于应用程序内部和应用程序之间的通信。为了防止应用程序模仿系统意图，Android限制了谁可以发送某些意图。所有Intent都通过ActivityManagerService（系统服务）发送，该服务强制执行此限制。 使用两种技术来限制系统意图的发送。某些Intent只能由具有适当权限的应用程序发送。 其他系统意图只能由UID与系统匹配的进程发送。不管它们拥有什么权限，应用程序都无法发送后一种类别的Intent，因为这些Intent必须源自系统进程。

应用程序可能还需要权限才能接收某些系统意图。操作系统使用标准的Android机制来限制其Intent收件人。应用程序（在这种情况下为OS）可以通过向Intent附加权限要求来限制谁可以接收Intent。

# 权限测试方法

Android的访问控制政策没有充分记录，但是该政策对于确定应用程序是否特权过多是必需的。为了解决这个缺点，我们根据经验确定了Android实施的访问控制策略。我们使用测试来构建权限图，该权限图标识Android API中每种方法所需的权限。特别是，我们修改了Android 2.2的权限验证机制，以记录发生的权限检查。然后，我们为API调用，内容提供者和意图生成了单元测试用例。 执行这些测试使我们能够观察与系统API交互所需的权限。一项核心挑战是建立单元测试，以获取所有平台资源的呼叫覆盖范围。

## 3.1 API

如2.2.1中所述，Android API为应用程序提供了一个库，该库包含公共，私有和隐藏的类和方法。私有类的集合包括用于系统服务的RPC存根。1所有这些类和方法都可以使用Java反射对应用程序进行访问，因此我们必须对其进行测试以识别权限检查。我们分三个阶段进行测试：针对反馈的测试；可定制的测试用例生成；手动验证。

### 3.1.1 定向反馈测试

在测试的第一阶段，我们使用了Randoop，这是一种针对Java的，面向反馈的，面向对象的自动化测试生成器[20，22]。Randoop将类列表作为输入，并从这些类中搜索可能的方法序列的空间。我们修改了Randoop，使其可以作为Android应用程序运行并记录其调用的每个方法。我们对Android的修改记录了Android权限验证机制检查的每个权限，这使我们可以推断出哪个API调用触发了权限检查。

Randoop搜索方法的空间以查找其返回值可用作其他方法的参数的方法。它维护有效的初始输入序列和参数的池，这些初始输入序列和参数最初是用原始值（例如int和String）播种的。Randoop通过从测试类的方法中随机选择一个方法，并从输入池中选择序列以填充该方法的参数来逐步构建测试序列。如果新序列是唯一的，则将执行它。成功完成的序列（即不产生异常）将添加到序列池中。Randoop的目标是全面覆盖测试空间。与同类技术[4,9,21]不同，Randoop不需要示例执行跟踪作为输入，从而使诸如API模糊测试之类的大规模测试更加易于管理。由于Randoop使用Java反射从提供的类列表中生成测试方法，因此它支持测试非公共方法。我们修改了Randoop以测试输入类的嵌套类。

局限性。Randoop的反馈制导空间探索受到它可以访问的对象和输入值的限制。如果Randoop无法在序列池中找到调用方法所需的正确类型的对象，则它将永远不会尝试调用该方法。Android API太大，无法一次测试所有相互依赖的类，因此实际上在序列池中没有许多对象可用。我们通过一起测试相关类（例如Account和AccountManager）并添加返回常见Android特定数据类型的种子序列来缓解此问题。不幸的是，这不足以为许多方法产生有效的输入参数。许多单例对象实例只能通过带有特定参数的API调用来创建；例如，可以通过使用参数“wifi”调用android.content.Context.getSystemService（String）来获得WifiManager实例。我们通过使用特定的原始常量和序列扩展输入池来解决此问题。另外，一些API调用期望内存地址存储参数的特定值，而我们无法大规模解决这些问题。

Randoop也不处理与输入参数无关的订购要求。在某些情况下，Android期望方法以非常特定的顺序彼此优先。Randoop仅生成序列链是为了为方法创建参数。它不能生成序列来满足非输入变量形式的依赖关系。进一步加重了这个问题，许多具有底层本机代码的Android方法如果被无序调用会产生分段错误，从而终止Randoop测试过程。

### 3.1.2 可定制的测试用例生成

Randoop的以反馈为导向的测试方法未能涵盖某些类型的方法。发生这种情况时，无法手动编辑其测试序列以控制序列顺序或建立方法前提条件。为了解决这些限制并提高覆盖率，我们构建了自己的测试生成工具。我们的工具接受方法签名列表作为输入，并为每种方法输出至少一个单元测试。它维护一个默认输入参数池，这些默认参数可以传递给要调用的方法。如果一个参数有多个值，那么我们的工具将为该方法创建多个单元测试。（当同一方法的多个参数具有多个可能的值时，将组合创建测试。）如果找不到合适的参数，它也会使用空值生成测试。由于我们的工具将测试用例的生成与执行分开，因此人工测试人员可以编辑由我们的工具生成的测试序列。如果测试失败，我们将手动调整方法调用的顺序，引入额外的代码以满足方法的先决条件，或者为失败的测试添加新的参数。

我们的测试生成工具比Randoop需要更多的人工，但是对于快速覆盖Randoop无法正确调用的方法来说，它是有效的。与手动编写测试用例相比，监督和编辑由我们的工具生成的一组生成的测试用例的工作量仍然要少得多。我们在大规模API测试中的经验是，反馈定向测试难以调用的方法经常会出现问题。 当人类测试人员能够编辑失败的序列时，可以正确调用这些方法。

### 3.1.3 手动验证

测试的前两个阶段生成API中每种方法执行的权限检查的映射。但是，这些结果包含三种类型的不一致。首先，由异步API调用引起的权限检查有时会错误地与后续API调用相关联。其次，方法的权限要求可能取决于参数，在这种情况下，我们会对该方法进行间歇性或不同的权限检查。第三，权限检查可以取决于API调用的顺序。为了识别和解决这些不一致之处，我们手动验证了测试的前两个阶段生成的权限图的正确性。

我们使用了可定制的测试生成工具来创建测试，以确认与权限图中每个API方法相关联的权限。我们仔细测试了测试用例的顺序和参数，以确保我们将权限检查与异步API调用正确匹配，并确定了权限检查的条件。在确认潜在的异步或依赖于订单的API调用的权限时，我们还为相关类中最初没有与权限检查相关联的相关方法创建了确认测试用例。我们运行每个测试用例，无论它们是否具有必需的权限，以标识具有多个或可替换权限要求的API调用。如果测试用例在未经许可的情况下引发安全异常，但在获得许可后成功，那么我们知道被测方法的许可权映射是正确的。

*测试Internet权限。*应用程序可以通过Android API访问Internet，但是其他包（例如java.net和org.apache）也提供Internet访问。为了确定哪些方法需要访问Internet，我们搜索了文档，并在Internet上搜索了建议访问Internet的所有方法。使用此列表，我们编写了测试用例，以确定哪些方法需要INTERNET权限。

## 3.2 Content Providers

我们的Content Provider测试应用程序对与Android系统和预安装的应用程序相关联的Content Provider URI执行查询、插入、更新和删除操作。我们从android.provider包中收集了URI列表，以确定要测试的内容提供商的核心集。我们还收集了在其他测试阶段发现的Content Provider URI。对于每个URI，我们尝试在没有任何权限的情况下执行每种类型的数据库操作。如果引发了安全异常，我们将记录所需的权限。我们添加并测试了权限的组合，以标识多个或可替代的权限要求。每个内容提供者都经过测试，直到不再为给定操作引发安全异常，这表明完成该操作所需的最小权限集。除了测试外，我们还检查了系统内容提供商的静态权限声明。

## 3.3 Intents

我们构建了一对应用程序来发送和接收Intent。Android文档没有提供可用系统Intent的单个完整列表，因此我们抓取了公共API来查找可能是Intent内容的字符串常量。我们在测试应用程序之间使用这些常量发送和接收Intent。为了测试接收系统广播意图所需的权限，我们通过发送和接收短信，发送和接收电话，连接和断开WiFi，连接和断开蓝牙设备等来触发系统广播。对于所有这些测试，我们记录是否进行了权限检查以及意图是否已成功交付或接收。

# 权限映射结果

我们对Android应用程序平台的测试产生了一个权限图，该权限图将权限要求与API调用，内容提供者和意图相关联。在本节中，我们将讨论API的涵盖范围，将结果与Android官方文档进行比较，并介绍Android API和权限映射的特征。

## 4.1 覆盖范围

Android API包含1665个类，共有16732个公共和私有方法。 通过两个阶段的测试，我们达到了Android API覆盖率的85％。 （如果在不产生异常的情况下执行该方法，则将其定义为被覆盖的方法；我们不衡量分支的覆盖范围。）Randoop的初始方法覆盖率为60％，分布在所有程序包中。 我们使用专有的测试生成工具补充了Randoop的覆盖范围，并通过至少一项权限检查来完成对属于类的方法的覆盖率接近100％。

API的未发现部分是由于本机调用和未在第一阶段进行权限检查的软件包的第二阶段测试所致。首先，当提供不正确的参数时，本机方法经常使应用程序崩溃，从而使其难以测试。 许多本机方法参数是整数，它们表示本机代码中对象的指针，因此很难提供正确的参数。 大约三分之一的未发现方法是本机调用。其次，我们决定对在Randoop测试阶段未显示权限检查的软件包省略补充测试。如果Randoop没有在包中触发至少一项权限检查，则我们不会在包中的类上添加更多测试。

## 4.2 与文档比较

清晰，完善的文档可促进正确使用权限和安全的编程习惯。 文档中的错误和遗漏可能导致错误的开发人员假设和特权。 Android的权限文档受到限制，这很可能是因为它们缺乏集中式访问控制策略。 我们的测试确定了1259个具有权限检查的API调用。 我们将此与Android 2.2文档进行了比较。

我们检索了Android 2.2文档，发现该文档指定了78种方法的权限要求。 该文档另外在几个类描述中列出了权限，但是尚不清楚类的哪些方法需要声明的权限。在文档中的78个受权限保护的API调用中，我们的测试表明6个API调用的文档不正确。我们不清楚文件或实现是否错误；如果文档正确，则这些差异可能是安全错误。

其中三个文档错误列出了与通过测试发现的权限不同的权限。在一个地方，文档声称一个API调用实际上受到较低权限的普通权限GET\_ACCOUNTS的访问，并且受到危险权限MANAGE\_ACCOUNTS的保护。另一个错误声称API调用需要ACCESS\_COARSE\_UPDATES权限，该权限不存在。结果，我们在x6.2中研究的900个应用程序中有5个请求此不存在的权限。第三个错误指出，当方法实际上受BLUETOOTH\_ADMIN保护时，该方法受BLUETOOTH许可保护。

|  |  |
| --- | --- |
| **Permission** | **Usage** |
| BLUETOOTH | 85 |
| BlUETOOTH\_ADMIN | 45 |
| READ\_CONTACTS | 38 |
| ACCESS\_NETWORK\_STATE | 24 |
| WAKE\_LOCK | 24 |
| ACCESS\_FINE\_LOCATION | 22 |
| WRITE\_SETTINGS | 21 |
| MODIFY\_AUDIO\_SETTINGS | 21 |
| ACCESS\_COARSE\_LOCATION | 18 |
| CHANGE\_WIFI\_STATE | 16 |

表1：Android的10个最常检查权限。

其他三个文档错误与具有多个权限要求的方法有关。 在一个错误中，文档声称一种方法需要一个许可，但是我们的测试表明需要两个许可。 对于最后两个错误，文档指出两个方法每个都需要一个许可权。 但是实际上，这两种方法都接受两种权限（即，它们是ORs）。

## 4.3 权限描述

基于我们的权限映射，我们描述了如何在整个API中分布权限检查。

### 4.3.1 API调用

我们检查了Android API，看看有多少方法和类进行了权限检查。我们提供权限检查的数量、未使用的权限、层次权限、权限粒度和类特征。

**权限检查数。**我们通过权限检查识别了1244个API调用，占所有API方法（包括隐藏和私有方法）的6.45%。其中，816是普通API类的方法，428是用于与系统服务通信的RPC存根的方法。我们在一个制造商添加的API的一个补充部分中另外识别了15个带有权限检查的API调用，总共1259个带有权限检查的API调用。表1提供了普通API最常见的检查权限的速率。

**签名/系统权限。**我们发现12%的普通API调用使用签名/系统权限进行保护，35%的RPC存根使用签名/系统权限进行保护。这有效地限制了这些API调用对预安装应用程序的使用。

**未使用的权限。**我们发现有些权限是由平台定义的，但从未在API中使用过。例如，BRICK许可从未被使用，尽管经常被引用为一个特别可怕的许可的例子。BRICK权限的唯一使用是死代码，它不能对设备造成损害。我们的测试发现，134个Android定义的权限中有15个未使用。对于在测试期间从未找到权限的每个情况，我们搜索源树以验证是否未使用该权限。在检查了几个设备之后，我们发现HTC和Samsung添加到API中以支持手机4G的自定义类使用了其中一个未使用的权限。

**分级权限。**许多权限的名称意味着它们之间存在层次关系。直观地说，我们期望更强大的权限应该可以替换与同一资源相关的较小权限。然而，我们没有发现有计划的等级制度的证据。我们的测试表明，BLUETOOTH\_ADMIN不能替代BLUETOOTH，WRITE\_CONTACTS也不能替代READ\_CONTACTS。同样，不能使用更改WIFI状态来代替访问WIFI状态。

只有一对权限具有层次关系：ACCESS\_COARSE\_LOCATION和ACCESS\_FINE\_LOCATION。每个接受COARSE权限的方法也接受FINE作为替代。我们发现只有一个例外，这可能是一个bug：电话管理员。TelephonyManager.listen() 支持ACCESS\_COARSE\_LOCATION或READ\_PHONE\_STATE权限，但不支持ACCESS\_FINE\_LOCATION权限。

**权限粒度。**如果将单个权限应用于不同的功能集，则请求对功能子集的权限的应用程序将对其余部分具有不必要的访问权限。Android的目标是在可能的情况下，通过将功能分成多个权限来防止这种情况，并且他们的方法已经被证明有利于平台安全。作为一个案例，我们研究了蓝牙功能的划分，因为蓝牙权限是检查最多的权限。

我们发现这两个蓝牙权限应用于6个大类。它们分为更改状态的方法（蓝牙管理）和获取设备信息的方法（蓝牙）。BluetoothAdapter类是使用Bluetooth权限的几个类之一，它适当地划分了大部分权限分配。然而，它有一些不一致之处。一个方法只返回信息，但需要BLUETOOTH\_ADMIN权限，另一个方法更改状态，但同时需要这两个权限。这种类型的不一致可能会导致开发人员混淆哪些类型的操作需要哪些权限。

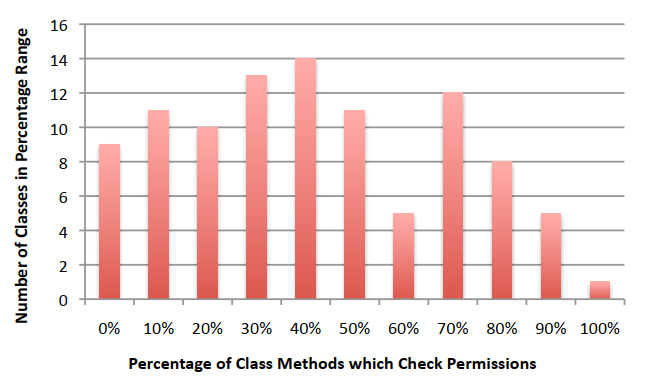
**阶级特征。**图2显示了每个类受保护的方法的百分比。我们最初预计，分布将是双峰的，大多数类完全或根本不受保护。然而，我们看到的是一系列的等级保护率。在这些类中，只有8个类需要权限来实例化对象，4个类只需要对象构造函数的权限。

图2：类的数量直方图，按需要权限的类的方法的百分比排序。所示数字代表

范围，即10%代表[10-20%]。我们只考虑至少有一个权限检查的类。

### 4.3.2 Content Providers and Intents

我们检查了内容提供者，以确定它们是否受权限保护。我们总共调查了62个内容提供商。我们发现有18个内容提供者不具有我们测试的任何方法（插入，查询，更新和删除）的权限。所有缺少权限的内容提供者都与content：// media内容URI相关联。

我们检查了意图通信，并测量了发送和接收意图是否需要权限。发送广播Intent时，非系统发送者禁止62个广播，发送Intent之前有6个需要许可，并且2个广播可以被广播，但系统接收者无法接收。广播接收方必须具有接收23个广播Intent的权限，其中有14个受蓝牙许可保护。发送意图以开始活动时，7条意图消息需要权限。 启动服务时，2个意图需要权限。

# 应用分析工具

我们构建了一个静态分析工具Stowaway，它可以分析Android应用程序并确定其可能需要的最大权限集。Stowaway分析了应用程序对API调用，Content Providers和Intents的使用，然后使用x3中内置的权限图来确定这些操作需要哪些权限。

Android平台的已编译应用程序包括可在Android Dalvik虚拟机上运行的Dalvik可执行文件（DEX）文件。我们使用公开可用的Dedexer工具来分解应用程序DEX。Stowaway的每个阶段都将分解后的DEX作为输入。

## 5.1 API 调用

Stowaway rst解析反汇编的DEX文件并识别对标准API方法的所有调用。Stowaway跟踪从An-droid类继承方法的应用程序定义的类，因此我们可以区分应用程序定义的方法和Android定义的继承方法的调用。我们使用启发式来处理Java响应和两个异常权限。

反射。Java反射是一个具有挑战性的问题。在Java中，可以使用Java反射地调用方法。lang.reflect.Method.invoke（）或java.lang.reflect。Constructor.newInstance（）。偷偷摸摸跟踪哪些Class对象和方法名称传播到反射调用。它执行流程敏感的过程内静态分析，并通过过程间分析将其扩展到2个方法调用的深度。在每个方法主体中，它跟踪每个String，StringBuilder，Class，Method，构造函数，Field和Object的值。我们还跟踪这些类型的静态成员变量的状态。我们确定将字符串和对象转换为Class类型的方法调用，以及将Class对象转换为Method，Constructor和Fields的方法调用。

我们还通过处理可能影响反射调用的方法和字段，将Android特定的启发式方法应用于反射解决方案。我们无法对整个Android和Java API的行为进行建模，但是我们会识别特殊情况。首先，Context.getSystemService（String）根据参数返回不同类型的对象。我们维护参数到返回对象类型的映射。其次，一些API类包含私有成员变量，这些变量包含对隐藏接口的引用。应用程序只能以反射方式访问这些成员变量，从而模糊了它们的类型信息。我们在成员变量及其类型之间创建了映射，并相应地传播了类型数据。如果应用程序在检索到某个成员变量后随后访问该方法，我们可以解析该成员变量的类型。

**互联网。**包含WebView的任何应用程序都必须具有Internet权限。WebView是一个用户界面组件，它允许应用程序将网站嵌入其UI中。WebView可以以编程方式实例化，也可以在XML文件中声明。Stowaway标识WebView的程序化实例。它还会反编译应用程序XML文件并解析它们以检测WebView声明。

**外部存储。** 如果应用程序要访问SD卡上存储的文件，则必须具有WRITE\_EXTERNAL\_STORAGE权限。 该权限未出现在我们的权限图中，因为（1）完全使用Linux权限强制执行，并且（2）可以与从库中访问SD卡的任何文件操作或API调用相关联。 我们通过在应用程序的字符串文字和XML文件中搜索包含sdcard的字符串来处理此权限； 如果找到，则假定需要WRITE\_EXTERNAL\_STORAGE。 此外，如果我们看到返回到SD卡路径的API调用（例如Environment.getExternalStorageDirectory（）），则假定需要此权限。

## 5.2 Content Providers

通过对URI执行数据库操作来访问内容提供者。 Stowaway收集了所有可用作内容提供者URI的字符串，并将这些字符串链接到内容提供者的权限要求。 内容提供者URI可以通过两种方式获得：

* + - 1. 可以将一个字符串或一组字符串传递到返回URI的方法中。 例如，API调用android。 net.Uri.parse（“ content：// browser / bookmarks”）返回用于访问浏览器书签的URI。 为了处理这种情况，Stowaway查找了所有以content：//。
      2. 该API提供了包含公共URI常量的Content Provider帮助器类。例如，android.provider.Browser.BOOKMARKS\_URI的值为content：// browser / bookmarks。Stowaway识别已知的URI常量，我们创建了一个从所有已知URI常量到其字符串值的映射。

我们工具的局限性在于我们无法判断应用程序使用URI执行哪些数据库操作。在Content Provider上执行操作的方式有很多，用户可以设置自己的查询字符串。为了解决这个问题，我们说应用程序可能需要与给定内容提供者URI上的任何操作相关的任何许可。这提供了使用特定内容提供程序可能需要的权限上限。

## 5.3 Intents

我们使用ComDroid 检测需要权限的Intent的发送和接收。ComDroid执行对流程敏感的过程内静态分析，并通过有限的过程间分析进行扩充，该过程在方法调用之后达到一个方法调用的深度。ComDroid跟踪Intent，寄存器，接收器（例如sendBroadcast）和应用程序组件的状态。当实例化一个Intent对象，将其作为方法参数传递或作为返回值获取时，ComDroid会跟踪从其源到其接收器的所有更改，并输出有关Intent的所有信息以及希望接收消息的所有组件。

Stowaway获取ComDroid的输出，并针对每个发送的Intent，检查是否需要许可才能发送该Intent。对于注册应用程序要接收的每个Intent，Stowaway会检查是否需要许可才能接收Intent。有时，ComDroid无法识别消息或意图的接收器。为了减轻这些情况，Stowaway在应用程序中所有字符串文字的列表中搜索受保护的Intent。

# 应用分析结果

我们将Stowaway应用到940个Android应用程序中，以识别过度特权的普遍性。具有不必要权限的应用程序违反了最小特权原则。特权过高损害了每个应用程序权限系统的好处：额外的权限不必要地限制用户随意接受危险的权限，并不必要地加剧应用程序漏洞。

Stowaway计算应用程序可能需要的最大Android权限集。我们将该设置与应用程序实际请求的权限进行比较。如果应用程序请求更多权限，则它具有特权。 我们全套的应用程序由964个Android 2.2应用程序组成。我们预留了24个随机选择的应用程序进行工具测试和培训，剩下940个用于分析。

## 6.1 手动分析

### 6.1.1 方法

我们从940个集合中随机选择了40个应用程序，并在其中运行了Stowaway。斯托沃威将18项申请确定为特权过高的。然后，我们手动分析了每个特权特权警告，将其归因于工具错误（即误报）或开发人员错误。由于三类故障，我们寻找误报：

* + - 1. Stowaway错过了需要权限的API，Content Provider或Intent操作。例如，当Stowaway无法解决反射调用的目标时，它会错过API调用。
      2. Stowaway可以正确识别API，Content Provider或Intent操作，但是我们的权限图缺少该平台资源的条目。
      3. 该应用程序将Intent发送到其他某个应用程序，并且收件人仅接受具有特定许可权的发件人的Intent。Stowaway无法检测到这种情况，因为我们无法确定其他非系统应用程序的许可要求。

我们检查了18个应用程序的字节码，以查找这三种错误中的任何一种。 如果我们发现功能可能与Stowaway认为不必要的许可有关，则我们手动编写其他测试用例以确认我们的许可权地图的准确性。 我们通过检查应用程序是否将Intents发送到预安装的或知名的应用程序来研究第三种类型的错误。 当我们确定警告不是误报时，我们试图确定为什么开发人员添加了不必要的权限。

我们还通过在修改后的Android版本中运行应用程序（在发生权限检查时将其记录下来）并与之交互来分析过特权警告。无法在运行时测试所有应用程序。例如，某些应用程序依赖自我们下载它们以来已移动或更改的服务器端资源。我们能够以此方式测试18个应用程序中的10个。在每种情况下，运行时测试都确认了我们的代码审查的结果。

### 6.1.2 错误报告

Stowaway确定了40个应用程序中的18个（占45％）具有42个不必要的权限。我们的手动审查确定了17个应用程序（42：5％）具有特权，共有39个不必要的权限。 这代表7％的误报率。

所有这三个错误警告都是由于我们权限图中的不完整所致。每个都是我们无法预期的特例。三个误报中的两个是由使用Runtime.exec来执行权限保护的shell命令的应用程序引起的。（例如，logcat命令执行READ\_LOGS权限检查。）第三个误报是由以下应用程序引起的：嵌入使用HTML5地理位置的网站，该网站需要位置许可。我们针对这些情况编写了测试用例，并更新了权限图。

在这套应用程序中的40个应用程序中，有4个包含至少一个反射调用，我们的静态分析工具无法解决或关闭这些反射调用。其中2个享有特权。这意味着具有至少一个未解决的反射调用的应用程序中有50％的特权过高，而其他应用程序的特权为42％。但是，样本量4太小，无法得出结论。我们调查了未解决的反省电话，并且不相信它们会导致误报。

## 6.2 自动化分析

我们在900个Android应用程序上运行了Stowaway。总体而言，Stowaway确定323个应用程序（占35：8％）具有不必要的权限。Stowaway无法解决某些应用程序的反射调用，这可能导致这些应用程序中更高的误报率。因此，我们将与其他应用程序分开讨论具有未解决的反射调用的应用程序。

### 6.2.1 完全处理反射的应用

Stowaway能够处理900个应用程序中的795个的所有反射调用，这意味着它应该已经标识了那些应用程序的所有API访问。Stowaway为795个应用程序中的32：7％生成了特权特权警告。表2显示了这些应用程序中10种最常见的不必要权限。

56％的超特权应用程序具有1个额外的权限，而94％的应用程序具有4个或更少的额外权限。尽管三分之一的应用程序享有特权，但每个应用程序的过度特权程度低表明开发人员正在尝试添加正确的权限，而不是任意请求大量不需要的权限。这支持了Android等安装时权限系统的潜在效力。

我们相信，Stowaway对于这些应用程序应产生与在x6.1中评估的40个一组相同的假阳性率。 如果我们假设手动分析得出的7％误报率适用于这些结果，那么795项申请中的30.4％确实享有特权。 实际上，由于以下原因，应用程序实际上可能比我们的工具所显示的特权更多。

无法访问的代码。 偷渡者不执行无效代码消除； 消除Android应用程序的死代码需要考虑到独特的Android生命周期和应用程序入口点。 此外，我们对Content Provider操作（x5.2）的过高估计可能会忽略一些特权。 我们没有量化Stowaway的误报率，我们将消除无效代码和改进Content Provider字符串跟踪留给以后的工作。

|  |  |
| --- | --- |
| **Permission** | **Usage** |
| ACCESS\_NETWORK\_STATE | 16% |
| READ\_PHONE\_STATE | 13% |
| ACCESS\_WIFI\_STATE | 8% |
| WRITE\_EXTERNAL\_STORAGE | 7% |
| CALL\_PHONE | 6% |
| ACCESS\_COARSE\_LOCATION | 6% |
| CAMERA | 6% |
| WRITE\_SETTINGS | 5% |
| ACCESS\_MOCK\_LOCATION | 5% |
| GET\_TASKS | 5% |

表2：10种最常见的不必要权限以及请求它们的过度特权应用程序的百分比。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Apps with Warnings | Total Apps | Rate |
| Reflection, failures | 56 | 105 | 53% |
| Reflection, no failures | 151 | 440 | 34% |
| No reflection | 109 | 355 | 31% |

表3：按反射状态，Stowaway发出超权限警告的速率。

### 6.2.2 Java反射的挑战

反射通常在Android应用程序中使用。在900个应用程序中，有545个（占61％）使用Java反射进行API调用。我们发现反射被用于许多目的，例如反序列化JSON和XML，调用隐藏或私有API调用以及处理名称在版本之间变化的API类。反射的普遍性表明，即使Android静态分析工具并非旨在用于混淆代码或恶意代码，对于Android静态分析工具来说也很重要。

在59％的使用反射的应用程序中，Stowaway能够完全解决反射呼叫的目标。我们使用两种技术处理了117个应用程序：消除了已知在应用程序中定义了反射调用的目标类的故障，以及手动检查和处理了21个非常流行的库中的故障。这使我们有105个具有反射功能的应用程序Stowaway无法解决或驳回的呼叫，占900项申请的12％。

Stowaway将105个应用程序中的53：3％确定为特权级别较高。 表3将其与没有未处理反射的应用程序发出警告的比率进行了比较。 对此差异有两种可能的解释：Stowaway在具有未解决的反射调用的应用程序中可能具有较高的误报率，或者以复杂方式使用Java反射的应用程序可能由于相关的特性而具有较高的实际超特权率。

我们怀疑这两个因素在未处理反射呼叫的应用程序中较高的特权特权警告率中都起作用。尽管我们的人工审查（x6.1）并未发现反射失败会导致误报，但随后对其他应用程序的审查却发现了一些由反射引起的错误警告。另一方面，开发人员错误可能随着与复杂的反射调用相关的复杂性而增加。

在Android应用程序中提高反射调用的分辨率是一个重要的开放问题。出现基于非静态环境变量的方法名称，直接生成Dalvik字节码，带有两个引用相同位置的指针的数组或存储在哈希表中的Method和Class对象时，Stowaway的反射分析将失败。Stowaway的方法主要是线性遍历，也遇到非线性控制流的问题，例如跳跃； 我们只处理出现在方法末尾的简单getos。我们还观察到了几个在一组类或方法上进行迭代的应用程序，测试每个元素以确定反射性调用哪个元素。如果测试了多个比较值并且在该块中未使用任何比较值，则Stowaway仅跟踪该块之外的最后一个比较值；该值可以为空。将来的工作也许可以使用动态分析来解决其中的一些问题。

## 6.3 常见的开发人员错误

在某些情况下，我们能够确定为什么开发人员要求不必要的权限。在这里，我们考虑了手动审核的40个应用程序和自动化分析的795个完全处理的应用程序中不同类型的开发人员错误的普遍性。

**权限名称。**开发人员有时会要求发出听起来与他们的应用程序功能相关的名称的权限，即使这些权限不是必需的。 例如，我们手动审核中的一个应用程序不必要地请求MOUNT\_UNMOUNT\_FILESYSTEMS权限来接收android.intent.action.MEDIA\_MOUNTED Intent。作为另一个示例，ACCESS\_NETWORK\_STATE和ACCESS\_WIFI\_STATE权限具有相似的名称，但是不同的类需要它们。开发人员经常成对地请求它们，即使只需要一个。在不必要地请求网络许可的应用程序中，有32％合法地需要WiFi许可。在不必要地请求WiFi许可的应用程序中，有71％合法地需要网络许可。

**代表。**一个应用程序可以向另一个代理应用程序发送一个Intent，要求代理执行操作。如果代理进行权限保护的API调用，则代理需要权限。但是，Intent的发送者没有。我们注意到，有许多应用程序实例要求他们代表

代表执行操作的权限。例如，一个应用程序要求Android Market安装另一个应用程序。发件人询问INSTALL\_PACKAGES，因为Market应用程序进行安装，所以不需要。

我们发现这种错误的广泛证据。在不必要地请求CAMERA许可的应用程序中，有81％会发送一个Intent来打开默认的Camera应用程序进行拍照。不必要地请求INTERNET的应用程序中有82％发送了在浏览器中打开URL的Intent。同样，不必要地请求CALL\_PHONE的应用程序中有44％会将Intent发送到默认的Phone Dialer应用程序。

**相关方法。** 如图2所示，大多数类包含权限保护和不受保护的方法的混合。我们发现应用程序使用不受保护的方法，但请求同一类中其他方法所需的权限。例如android.provider。Settings.Secure是API中用于访问Settings Content Provider的便捷类。该类包括设置器和获取器。设置器需要WRITE\_SETTINGS权限，但获取器则不需要。我们手动检查的两个应用程序仅使用吸气剂，但请求WRITE\_SETTINGS权限。

**复制和粘贴。**热门留言板包含Android代码段和有关权限要求的建议。有时，此信息不准确，复制该信息的开发人员将使他们的应用程序拥有特权。例如，我们手动检查的应用程序之一注册以接收android.net.wifi.STATE\_CHANGE意向并请求ACCESS\_WIFI\_STATE权限。截至2011年5月，该Intent的第三高Google搜索结果包含错误的主张，即它需要该许可权。

**不建议使用的权限。** 在旧版Android中，可能需要Android 2.2中不必要的权限。因此，旧的或向后兼容的应用程序可能具有额外的权限。但是，开发人员也可能不小心使用了这些权限，因为他们已阅读了过时的资料。8％的特权应用程序请求ACCESS\_GPS或ACCESS\_LOCATION，它们在2008年已弃用。在这些应用程序中，除一个之外，所有应用程序均指定其受支持的最低API版本高于该版本。

包含这些权限的最新版本。

**测试工件。**开发人员可能会在测试过程中添加一个权限，然后在删除测试代码时忘记将其删除。例如，ACCESS\_MOCK\_LOCATION通常仅用于测试，但可以在已发布的应用程序中找到。数据集中所有不必要地包含ACCESS\_MOCK\_LOCATION权限的应用程序也都包含真实位置权限。

**签名/系统权限。**我们发现9％的特权过多的应用程序请求不需要的Signature或SignatureOrSystem权限。Android的标准版本将默默拒绝将那些权限授予未由设备制造商签名的应用程序。权限被错误地请求，或者开发人员发现相关代码在标准手机上不起作用后删除了相关代码。

我们可以将许多特权实例归因于开发人员对许可系统的困惑。 可以通过改进的API文档解决权限名称，相关方法，代理和不赞成使用的权限的混淆。为避免由于相关方法而产生的超权限，建议您按方法（而不是按类）列出权限要求。通过阐明权限和预安装的系统应用程序之间的关系，可以减少对代理人的混淆。

尽管我们可以将许多不必要的权限归因于错误，但仍有一些开发人员有意地请求额外的权限。 激励开发人员要求不必要的权限，因为如果应用程序的更新版本请求更多权限，则应用程序将不会收到自动更新。

# 相关工作

**Android权限。**以前对Android应用程序的研究在了解权限使用方面受到了限制。我们的权限图可用于大大增加应用程序分析的范围。Enck等。将Fortify的Java静态分析工具应用于反编译的应用程序；他们研究API的用法。但是，他们仅限于研究应用程序对少量权限和API调用的使用。在最近的一项研究中，Felt等人。手动将一小部分Android应用程序分类为是否特权过度，但是它们受到Android文档的限制。麒麟在安装过程中读取应用程序许可要求，并根据一组安全规则进行检查。它们仅依赖于开发人员权限请求，而不是检查应用程序是否使用权限或如何使用权限。Barrera等。检查1100个Android应用程序的权限要求，并使用自组织映射来可视化具有相似特征的应用程序中使用了哪些权限。他们的工作还依赖于应用程序请求的权限。

维达斯等。提供对应用程序源代码执行特权分析的工具。通过使用我们的权限图可以改进他们的工具；他们基于有限的Android文档。我们的静态分析工具还可以执行更复杂的应用程序分析。与他们的Eclipse插件不同，Stowaway尝试处理反射调用，Content Providers和Intent。

在并发工作中，Gibler等。将静态分析应用于Android API以查找权限检查。它们的权限映射包括系统进程内无法跨RPC边界访问的内部方法，我们将其排除在外，因为应用程序无法访问它们。与我们的动态方法不同，它们的静态分析可能会误报，会丢失本机代码中的权限检查，并且会丢失特定于Android的控制流。

**Java测试。**Randoop不是唯一的Java单元测试生成工具。诸如Eclat，Palulu和JCrasher之类的工具的工作原理相似，但需要示例执行作为输入。考虑到Android API的大小，构建这样的示例执行将是一个挑战。 增强的JUnit通过将构造函数链接到某个固定深度来生成测试。但是，它不使用子类型提供实例，而是依靠字节码作为输入。Korat需要正式的方法规范规范作为输入，这对于Android API的事后测试是不可行的。

**Java反射。**处理Java反射对于开发完善的程序分析是必需的。但是，解决反射性呼叫是开放研究的领域。Livshits等。创建了一种静态算法，通过跟踪传递给反射的字符串常数来近似反射目标。当反射调用依赖于用户输入或环境变量时，他们的方法就不够用了。我们使用相同的方法并遭受相同的限制。他们通过开发人员注释来改善结果，这对于我们的领域而言并不可行。一种更高级的技术将静态分析与有关Java程序环境的信息相结合，以解决反射问题。但是，仅当程序在与原始评估相同的环境中执行时，它们的结果才是正确的。即使进行了修改，它们也只能解决Java 1.4 API中74％的反射调用。我们不主张在解决Java反射方面改进现有技术；相反，我们专注于特定于域的启发式方法，以了解如何在Android应用程序中使用反射。我们是第一个讨论Android应用程序反射的人。

# 结论

在本文中，我们开发了用于检测Android应用程序中超权限的工具。我们将自动化测试技术应用于Android 2.2，以确定调用每种API方法所需的权限。我们的工具Stowaway会生成应用程序所需的最大权限集，并将它们与实际请求的权限集进行比较。目前，Stowaway无法处理一些复杂的反射调用，并且我们将Java反射确定为Android静态分析工具的重要开放问题。我们将Stowaway应用到940个Android应用程序中，发现其中约有三分之一具有特权。我们的结果表明，应用程序通常仅受到少数权限的特权，而许多额外的权限可归因于开发人员的困惑。这表明开发人员试图为其应用程序获取最少的特权，但由于API文档错误和缺乏开发人员的了解而未能达到要求。

# 致谢

感谢Royce Cheng-Yue和Kathryn Lingel在测试API和内容提供程序方面的帮助。 NSF赠款CCF-0424422、0311808、0832943、0444482、0942694，来自Google的礼物以及AFOSR赠款FA9550-08-1-0352下的MURI程序部分支持了这项工作。该材料还基于NSF研究生研究奖学金的支持。这里表达的任何观点，发现，结论或建议均为作者的观点，不一定反映NSF的观点。