CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 36 卷第 11 期 2014 年 11 月
Vol. 36, No. 11, Nov. 2014

文章编号:1007-130X(2014)11-2074-13

Android 应用程序权限自动裁剪系统*

白小龙

(清华大学计算机系,北京 100084)

摘　要:Android 系统使用权限机制对应用程序进行控制,即应用程序需要使用哪些系统资源就必须提前声明相应的权限。为了确保安全性和可靠性,应用程序声明权限时应该满足最小特权原则,即只声明其所需要使用到的最少权限,但现实中有很多应用存在权限过度声明的现象,给用户带来安全隐患。提出了一种 Android 应用程序权限自动裁剪系统 PTailor,通过对 Android 应用程序安装文件(APK 文件)进行分析和修改,使其满足最小特权原则。PTailor 首先从 APK 文件中提取程序所调用的所有系统 API,并在预先生成的 API 权限映射表中查找该 API 所对应的系统权限,从而得到应用程序实际使用到的最少权限列表。然后根据该权限列表对程序的权限声明文件进行修改,裁剪掉已声明但未使用的权限。最后将裁剪过的权限声明文件与程序的其他部分重新合并成新的 APK 文件,新的 APK 文件中除了所声明权限满足最小特权原则外,其结构和语义都没有发生改变。使用 PTailor 对现实中的 1 246 个 Android 应用进行权限裁剪实验,实验结果表明,PTailor 能够在很短的时间内完成权限分析和裁剪,而且大多数被裁剪的程序都能够正确运行。

关键词:Android 应用程序;最小特权原则;权限过度声明;权限自动裁剪

中图分类号:TP316　　**文献标志码**:A

doi:10.3969/j.issn.1007-130X.2014.11.005

A system for automatically tailoring Android applications' permissions

BAI Xiao-long

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: Android uses the permission system to control application access. In the permission system, applications have to declare relevant permissions before they access some system resources. To be secure and trusted, applications should follow the principle of least privilege. However, in reality, many applications do not follow this principle, which may bring security threats. To solve this problem, we design and implement a novel system for automatically tailoring Android applications' permissions, called PTailor. PTailor analyzes and modifies the Android application installation file (APK file) so as to make it follow the principle of least privilege. Firstly, PTailor extracts the system API calls from the APK file and gets the API's corresponding required permissions from a predefined API-to-permissions map. In this way, PTailor can get the shortest permission list that this application really requires. PTailor uses this permission list to match the application's permission declaration file and removes those unused permissions. At last, the modified permission declaration file and the original code file are zipped to a new APK file that follows the principle of least privilege without changing its structure and semantics. PTailor is used to process 1246 Android applications in order to evaluate its performance. The experimental results show that APK files can be processed in a short time and PTailor has little influence on most tai-

* 收稿日期:2014-06-11;修回日期:2014-08-16
基金项目:国家 863 高技术研究发展计划资助项目(2011AA01A203)
通信地址:100084 北京市海淀区清华大学西主楼 1 区 417
Address: Room 1-417, West Main Building, Tsinghua University, Haidian District, Beijing 100084, P. R. China

lored applications.

Key words: Android application; the principle of least privilege; overprivileged; automatically tailor permissions

1 引言

近年来，随着智能手机的普及率不断提高，手机安全问题越来越引起人们的重视。越来越多的手机应用程序给用户带来便利的同时，也存储着越来越多的用户隐私数据，因此，手机应用程序的安全性和可靠性尤为重要。

智能手机中，Android 手机占有了巨大的市场份额。根据市场调研机构 Strategy Analytics 的最新研究报告[1]，2013 年 Android 手机占全球智能手机市场份额高达 79%，增幅达 62%。但是，由于系统开源性和应用市场开放性，Android 平台极易遭到攻击。而且由于 Android 应用程序的编写没有明确的规范，这使得 Android 应用程序的安全性和可靠性良莠不齐。

Android 系统使用一种权限机制来对应用程序进行访问控制，应用程序想要通过系统提供的 API 进行某种操作或使用某种资源，就必须具有与该 API 相对应的权限。这些权限必须声明在程序的 AndroidManifest. xml 文件中，该文件作为应用的重要组成部分在应用被安装时由系统进行检查并提醒用户该应用具体声明了哪些权限。应用程序在进行操作或使用资源时，Android 系统会检查其是否已声明过相应的权限。例如，从图 1 的代码段中可以看出，应用程序需要使用 TelephonyManager 的 getDeviceId()函数来获取设备编号，并使用 SmsManager 的 sendTextMessage()函数来发送短信，则该应用程序必须在其 AndroidManifest. xml 文件中声明 READ_PHONE_STATE 和 SEND_SMS 权限(如图 2 所示)，而系统会在该应用被安装时提醒用户该应用将读取手机状态和 ID 并发送短信(如图 3 所示)。

```
1. TelephonyManager tm=(TelephonyManager)
   getSystemService(TELEPHONY_SERVICE);
2. String deviceId=tm.getDeviceId();
3. SmsManager smsManager=SmsManager.getDefault();
4. smsManager.sendTextMessage("10010", null, deviceId,
   null, null);
```

Figure 1 Code snippet of using system resource

图 1 应用程序使用系统资源代码段

```
1. 〈uses-permission android:name="android.permission.
READ_PHONE_STATE"/〉
2. 〈uses-permission android:name="android.permission.
SEND_SMS"/〉
```

Figure 2 Xml segment of permission declaration

图 2 应用程序权限声明 xml 文档段

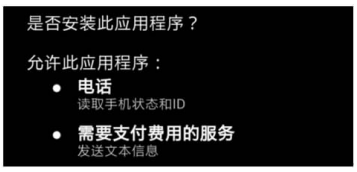

Figure 3 Screenshot of permission notification while installation

图 3 应用程序安装权限提醒截图

开发者在编写 Android 应用程序时应该遵循最小特权原则，即进行哪些操作或者使用哪些资源，就只声明与这些操作和资源相关的权限，而不应该声明过多不使用的权限。但是，很多开发者并没有遵循这一原则，造成了权限过度声明，文献[2]分析了 940 个 Android 应用，发现有 323 个声明了不使用的过多权限。过度声明的权限不仅会给用户带来误解，使用户对程序的可靠性和个人隐私的保密性产生怀疑，而且会带来安全隐患。当权限过度声明的应用程序存在 bug 时，这些程序可能会被其他恶意程序利用，从而对用户隐私或手机安全造成威胁。导致 Android 应用程序权限过度声明的主要原因有以下几点：

(1) Android 文档不完善。Android 文档未给出每个 API 所需的权限，有些文档给出的 API 权限信息甚至是错误的。文献[2]对 Android 2.2 进行分析发现，一共有 1 259 个 API 需要使用权限，但是，Android 文档只描述了其中的 78 个，其中有 6 个 API 所描述的权限与其实际所使用到的权限不符。

(2) 权限名称很接近，容易混淆。例如 ACCESS_NETWORK_STATE 和 ACCESS_WIFI_STATE，从名称看两者都是访问网络的状态，但实际上两者被用于不同的 API 中，程序员由于不清楚两者的具体含义，很容易同时声明这两个权限。

(3) Android 允许应用程序请求其他应用程序进行某些操作，这些被请求的应用需要具有相关的权限，但进行请求的应用并不需要这一权限。例

如,应用可以请求浏览器打开某一个URL,这时该应用并不需要INTERNET权限,因为访问网络的行为并不是由该应用完成的。有些开发者并不了解这点,导致了声明不必要的权限。

虽然有很多研究工作分析了应用程序权限过度声明现象,但还没有研究提出如何自动地处理和控制过度声明的权限。针对权限过度声明这一问题,本文提出了一种Android应用程序权限自动裁剪系统,称为PTailor(Permission Tailor)。PTailor系统通过检查Android应用程序安装文件(APK文件),提取程序中所有API调用,分析这些API所需的对应权限,获得程序所使用的最少权限列表,并通过该列表对应用程序所声明的权限列表进行裁剪,删除掉那些已声明但未使用的权限,从而使程序满足最小特权原则。且要求系统的裁剪处理过程能够在很短的时间内完成,并不会对大多数程序的正确运行产生影响,使其能够适用于对大量程序进行自动分析和裁剪,提高程序安全性和可靠性。

本文的主要贡献和创新点如下:

(1)提出了一种Android应用程序权限自动裁剪系统PTailor,PTailor通过对Android应用程序文件进行分析和修改使其满足最小特权原则。并通过实验对PTailor的权限裁剪性能、裁剪后可用性进行了评测。

(2)通过实验结果,对Android应用程序权限过度声明现象和发展趋势进行了分析,并对以往关于API权限分析工作的完整性进行了分析。

本文的具体内容组织如下:第2节介绍国内外与Android权限相关的研究现状;第3节将对Android权限等背景知识进行介绍;第4节重点介绍PTailor系统的组成模块,并对各模块的具体功能和算法进行详细的阐述;第5节阐述了实验设计以及结果分析,从多方面对PTailor系统进行了评测;第6节对PTailor设计和实验中的一些问题进行讨论,并对未来的工作进行展望;第7节对全文进行总结。

2 相关工作

Android系统采用权限机制对应用程序进行管理和控制,应用在安装时必须声明与其所使用到的资源相关的权限,且只能使用所声明过的权限范围内的资源。这一机制简单但控制能力有限,而且安装时声明权限的方式无论对于开发者还是用户都具有一定的局限性。国内外有很多学者针对这一权限管理机制的使用现状进行分析或者提出更加有效的管理方式。

2.1 API权限映射分析相关工作

由于Android文档对于大多数API所需权限没有明确的解释,导致了很多权限过度声明现象的发生。因此,研究API与其所需权限的映射,并分析现有应用程序中权限过度声明情况很有必要。

Felt A P等人[2]通过单元测试的方式分析了Android 2.2.1中的API权限映射关系,发现了1 259个API在使用时需要声明权限,但是,Android 2.2的文档中只给出了78个API的权限使用说明。同时,他们提出了Stowaway工具,用于检查应用程序的权限过度声明情况。并使用Stowaway对940个Android应用进行了分析,发现约三分之一(323)的应用存在权限声明但未使用的情况,并分析出产生这种现象的主要原因是Android文档的不完整。在此项研究基础上,Felt A P等人还对权限系统的设计提出了一些指导意见[3]。

Au K W Y等人[4]提出PScout系统,通过对Android系统权限检查部分源码进行静态分析的方式分析了API与其权限的映射关系,并分析了四个版本的Android系统中API与其权限的对应情况,包括隐藏的系统API和显式的通用API。同时,他们还通过模糊测试的方式对1 260个应用程序进行了权限声明检查,发现543个应用声明了额外未使用的权限。

相似地,Bartel A等人[5]提出了COPES系统,使用基于call graph的静态分析从Android系统框架中提取出函数调用与所需权限的对应关系。对Android 2.2进行分析,发现共有9 562个函数需要声明权限。但是,不同于PScout的是,该工作只分析了API函数调用中的权限,没有考虑到Intent和Content Provider等其他情况下所需的权限。

Vidas T等人[6]从Android文档中提取API权限映射,但由于Android文档的API权限信息非常不完善,所以该研究工作所获得的权限规范也不完整。

Wei X等人[7]使用Stowaway工具对237个应用的1 703个版本的权限声明演化过程进行了分析,发现19.6%在新版本中存在权限过度声明现象,25.2%在原本的版本中就已经过度声明。总体上呈现出权限过度声明的应用程序逐渐增多的

趋势。

Yang Bo 等人[8]提出了一种基于数据流分析的静态检测工具 Brox，用于检测 Android 应用程序是否声明了过多的权限。

这些研究工作虽然都对 Android 系统中 API 与其权限的映射关系进行了分析，或分析了应用程序的权限过度声明情况，但是都没有运用这一映射关系对应用程序的权限过度声明进行控制，也没有对过度声明的权限进行裁剪，无法确认 API 权限映射以及应用程序权限过度声明分析结果的有效性。

2.2 Android 权限其他相关研究

2.2.1 应用程序的权限特征分析

分析不同程序的权限特征可以帮助人们发现程序的潜在安全威胁。Enck W 等人[9]提出了 Kirin 系统，从应用程序中提取具有潜在安全威胁的权限组合，用于向用户提供安全建议。Barrera D 等人[10]对 1 100 个 Android 应用程序进行分类分析，使用自组织映像网络算法对不同功能类别应用程序所声明权限的特征进行分析。Peng H 等人[11]根据应用程序的权限组合使用概率生成模型对程序的危险指数进行评分。Pearce P 等人[12]分析了应用程序中广告所需的额外权限，并提出 AdDroid 架构用于将广告与应用程序主体分离，使得应用程序不需要为其广告额外声明权限。Zhang Y 等人[13]提出了 VetDroid 系统用于刻画程序的权限使用行为特征，从而更好地帮助安全分析师分析程序的内部敏感行为。Yang Huan 等人[14]使用包括权限信息在内的多种特征对 Android 应用程序的恶意行为进行检测。Zhang Rui 等人[15]基于 Android 应用程序权限的相关性特征对 Android 恶意软件进行聚类分析。

2.2.2 权限使用控制或提醒相关研究

让用户了解或控制程序正在使用的权限对于保护用户隐私很有帮助。Nauman M 等人[16]提出了 Apex 架构以及 Bao Ke-jin 等人[17]提出的权限管理模型，可以使用户选择将哪些权限赋予应用程序，从而起到自主访问控制的作用。Xu R 等人[18]提出的 Aurasium 系统可以在不修改 Android 源码或获得 Root 权限的情况下对应用程序的 API 使用请求进行用户提醒和访问控制。Livshits B 等人[19]在 Windows Phone 上添加了系统资源访问提醒，使得当应用程序访问某些系统资源时用户会得到相应的提醒，从而选择允许或拒绝该访问。

2.2.3 应用程序访问控制相关研究

Android 基于权限机制的访问控制方式虽然简单但是控制能力有限，很多学者致力于寻找更加有效的访问控制方式。由于 Android 系统是基于 Linux 操作系统的中间层系统，Smalley S 等人[20]基于 Linux 原有的一种强制访问控制机制——安全强化 Linux(SELinux)提出 SEAndroid，用于在 Linux 内核以及 Android 中间层框架上实现强制访问控制 MAC(Mandatory Access Control)。该技术现已被 Android 主流版本所吸纳。Bugiel S 等人[21]提出 FlaskDroid 系统，借助于开发者所定义的策略语言，实现灵活而细粒度的强制访问控制。Bugiel S 等人[22]还提出过基于 TOMOYO Linux 强制访问机制实现对 Android 应用进行强制访问控制的 TrustDroid。Ongtang M 等人[23]提出的 Saint 系统可以根据策略对应用程序安装时的权限授予和运行时的权限使用进行控制。Tang Wei[24]提出了基于安全距离的权限机制扩展方案，并提出了基于上下文的运行时检测方案。

3 背景介绍

3.1 Android 权限机制

Android 对系统 API 进行权限检查主要有三种情况：函数调用、Intent 和 Content Provider。

函数调用就是直接调用系统 API 函数从而使用某些资源，例如使用 SmsManager 类中的 sendTextMessage()函数来发送短信。

Intent 是 Android 系统中进程间通信的主要方式，即一个程序如果需要另一个程序完成某个工作，就需要向该程序发送一个具有特定 Action 参数的 Intent。而请求一些系统程序时则需要具有相应的权限才能发出 Intent。例如，应用请求使用拨号程序拨打电话，需要向拨号程序发送具有 android.intent.action.CALL 参数的 Intent，则该应用需要声明 android.permission.CALL_PHONE 权限。

Content Provider 存储着一些数据信息，并向其他程序开放用于获取相关的数据。应用程序如果需要获取 Content Provider 中的数据，则需要通过不同的 URL schema 进行请求，对于系统提供的某些 Content Provider，应用程序进行请求时必须具有相应的权限。例如，当应用程序使用 schema 为 content://sms 的 URL 请求读取设备中的短信

列表时就需要具有 android.permission.READ_SMS 权限。

3.2 APK 文件

Android 应用程序使用 Java 语言编写并编译成一种特殊的字节码文件,称为 Dalvik Bytecode,所有 Java Class 的 Dalvik Bytecode 合并在一起组成一个 dex 文件。除了 dex 文件,应用还需要在一个 AndroidManifest.xml 文件中声明应用程序的组成部分以及所使用到的权限。这个 AndroidManifest.xml 文件与 dex 文件以及其他一些所需要使用到的资源文件通过 JDK(Java Development Kit)中的 jar 工具打包并通过 jarsigner 工具使用开发者私钥进行签名,从而形成了 Android 应用程序包文件(简称 APK 文件),APK 文件即为应用程序的安装文件。

4 PTailor 系统

本文提出了一种 Android 应用程序权限自动裁剪的系统 PTailor,PTailor 主要采用静态分析技术,对 Android APK 文件进行分析和修改。PTailor 由五个部分组成,包括 API 权限映射表生成模块、APK 分解模块、API 提取模块、manifest 裁剪模块和 APK 合并模块。PTailor 系统架构如图 4 所示,对 APK 文件的分析和修改的流程如图 5 所示。

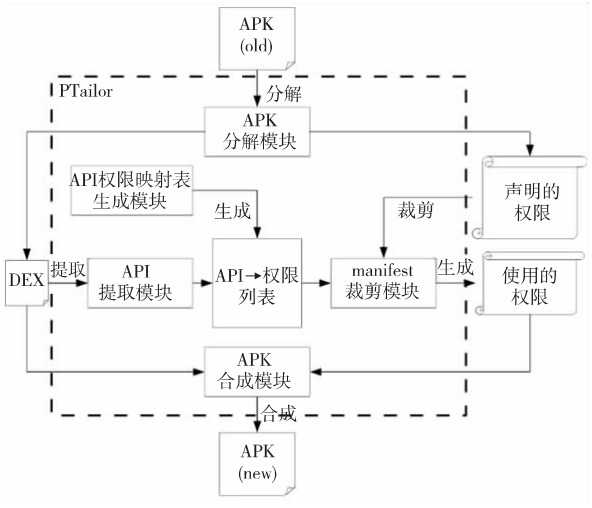

Figure 4 Architecture of PTailor

图 4 PTailor 系统架构图

4.1 API 权限映射表生成模块

API 权限映射表生成模块负责读取 API 权限映射数据文件,生成 API 与其权限的映射表,属于系统的准备工作。不同于其他模块对每个 APK 文件都要运行一次,该模块对所有需要处理的 APK 文件只需要运行一次即可。

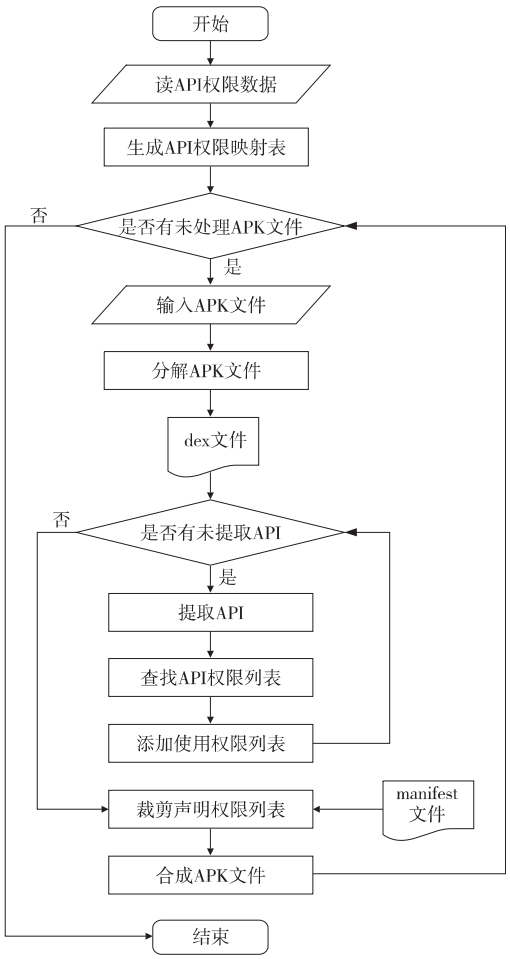

Figure 5 Workflow of PTailor

图 5 PTailor 对 APK 文件分析和修改流程图

4.1.1 API 权限映射数据源

本文中,我们使用 PScout[4] 中得到的 Android 4.1.1 API 权限对应结果数据作为 API 权限映射数据源。对应于 Android 权限检查机制的三种情况,这一数据源包括 API 函数调用与其所需权限对应数据、Intent Action 与其所需权限对应数据、Content Provider URL schema 与其所需权限对应数据。PTailor 同样可以使用其他数据作为其 API 权限映射源,也可以由用户自定义 API 权限的映射关系,但 PScout 的 API 权限映射关系是现阶段最为完整的,故本文目前采用该数据。

4.1.2 API 权限映射表数据结构

PTailor 使用散列表的数据结构存储 API 与其权限的映射关系,由于有些 API 会对应多个权限,所以需要使用单个 key 对应多个 value 的 Multimap 数据结构,以 API 为 key,以其所需权限为 value。这个 API 与权限的映射表 APM(API Per-

mission Map)用于在 API 提取模块中获得所提取的 API 所对应的权限。

4.2 APK 分解模块

APK 分解模块的主要工作是对 Android 应用程序 APK 文件进行解压缩，从而获得 dex 文件和 AndroidManifest.xml 文件。这两个文件分别用于 API 提取模块和 manifest 裁剪模块当中。API 提取模块通过遍历 dex 文件来检查所有 API 调用。manifest 裁剪模块通过 API 提取模块中所得到的实际使用的权限列表对 AndroidManifest.xml 文件所声明的权限进行修改，裁剪掉那些已声明但未使用的权限。经过裁剪的 AndroidManifest.xml 文件与原始的 dex 文件最后一起通过 APK 合成模块合成新的无权限过度声明的 APK 文件。

4.3 API 提取模块

API 提取模块是 PTailor 的核心组成部分。其主要作用是从 dex 文件中提取所有使用到的权限。在 3.1 节中已经介绍过，由于 Android 对 API 的权限检查包括函数调用、Intent 和 Content Provider 三种情况，所以 PTailor 的 API 提取模块需要对这三种情况中所使用到的权限进行分别提取。相对应地，API 提取模块共分为三个子模块，即函数调用提取模块、Intent 提取模块和 Content Provider 提取模块。这三个子模块都以 APK 分解模块中得到的 dex 文件和 API 权限映射表生成模块中产生的 API 权限映射表 APM 作为输入，输出为应用程序实际使用到的最小权限列表。

4.3.1 API 函数调用提取模块

API 函数调用提取模块的主要工作是提取程序中的所有函数调用，并在 API 权限映射表中查找所调用函数对应的权限，将查找到的权限添加到输出的已使用权限列表中。具体算法如算法 1 所示。

算法 1 API 函数调用权限提取算法

输入：Dalvik bytecode 文件 F，API 权限映射表 APM；

输出：已使用的权限列表 L。

```
1.  FOR each Class C in F DO
2.    FOR each Method M in C DO
3.      FOR each Instruction I in M DO
4.        θ ← GetOpcode ( I )
5.        IF θ = invoke THEN
6.          λ ← GetInvokedMethodWithClass ( I )
7.          IF λ is in APM THEN
8.            P ← GetPermissionsByAPI ( APM, λ )
9.            FOR each Permission p in P DO
10.             AddToList ( L, p )
11.           END
12.           ELSE
13.            α ← GetDeclaringClassName( λ )
14.            β ← GetMethodName ( λ )
15.            WHILE α is not NULL DO
16.              IF (α:β ) is in APM THEN
17.                P←GetPermissionsByAPI(APM, λ)
18.                FOR each Permission p in P DO
19.                AddToList ( L, p )
20.                END
21.                BREAK
22.              ELSE
23.                α ← GetSuperClass( α )
24.              END
25.            END
26.          END
27.        END
28.      END
29.    END
30.  END
```

算法 1 的输入为 APK 分解模块中所获得的 dex 文件 F 以及 API 权限映射表生成模块中所获得的 API 权限映射表 APM，输出为应用程序所使用到的权限列表 L。Android 应用程序是由多个 Java Class 组成的，而每个 Class 又是由多个 Java Method 组成的，每个 Method 中包含多个 Dalvik 指令(Instruction)，而只有 invoke 指令是函数调用指令，用于调用 API 函数。因此，算法 1 的第 1～第 3 行中，API 函数调用提取模块遍历每个 Class 的每个 Method 中的每个 Instruction，并在第 4、第 5 行检查这个 Instruction 是不是 invoke 指令。如果是，则通过 GetInvokedMethodWithClass()获取 invoke 指令所调用的函数 λ，λ 中包括该函数的名称、参数以及所属类。算法在第 7 行判断 API 权限映射表 APM 中是否有函数 λ 与其权限的映射，如果有，则在第 8～第 11 行中，将 APM 中 λ 所对应的所有权限加入到输出的已使用权限列表 L 中。为了保证 L 为最小权限列表，AddToList()对于同一个权限只会添加一次。

如果 APM 中没有函数 λ 的权限映射，则算法会在第 13～第 24 行检查 λ 是否有可能继承自 APM 中的某个 API。例如，使用类 X 中的 Z 函

数需要具有 P 权限,API 权限映射表中包含$(X:Z)\rightarrow P$的映射,但应用程序并没有直接使用类 X 中的 Z 函数,而是通过类 Y 继承类 X,并调用 Y 中的 Z 函数。虽然 Y 并没有对 Z 进行重写,且调用类 Y 中的 Z 函数同样需要 P 权限,但在 APM 中不包含$(Y:Z)\rightarrow P$的映射。为了处理这种情况,算法在第 13、第 14 行分别提取函数 λ 的所属类 α 和函数名称 β,在第 23 行回溯类 α 的继承关系链,并在第 16 行检查 APM 中是否含有$(\alpha:\beta)$的权限映射,如果没有,则在第 23 行继续回溯 α,如果有,则在第 17～第 20 行将 APM 中检查到的权限列表加入到 L 中。API 函数调用提取模块回溯函数的继承关系链的目的是防止由于某些应用通过继承系统服务的方式访问系统资源而导致的漏报。

4.3.2 Intent 和 Content Provider 提取模块

Intent 提取模块和 Content Provider 提取模块分别用于提取应用程序通过 Intent 方式和 Content Provider 方式获取系统资源时所需要的权限。

Intent 提取模块提取程序发出 Intent 请求时的 Action 参数,并在 API 权限映射表中查找这些 Action 参数所对应的权限,添加到已使用权限列表中。

Content Provider 提取模块提取程序发出的 URL 请求的 schema,并在 API 权限映射表中查找这些 schema 所对应的权限,添加到已使用权限列表中。

Action 参数和 URL schema 都是字符串类型,因此 Intent 和 Content Provider 提取模块查找这两个参数的方式是查找 dex 文件中是否有相应的字符串。

4.4 manifest 裁剪模块

经过 API 提取模块获得应用程序实际使用的权限列表之后,PTailor 通过 manifest 裁剪模块对声明了权限的 AndroidManifest.xml 文件进行修改,裁剪掉已声明但未使用的那些权限,从而达到最小特权原则。由于只修改 manifest 文件中权限声明部分,且只删除未使用过的权限声明,所以 PTailor 的 manifest 裁剪模块不会修改应用程序的结构。

4.5 APK 合并模块

APK 合并模块执行与 APK 分解模块相反的操作。经过 manifest 裁剪模块裁剪后的 AndroidManifest.xml 文件与 APK 分解模块中所得到的 dex 文件以及分解出的其他一些资源文件一起经过 APK 合并模块,重新合并成 APK 文件。重新合成的 APK 文件除了其 manifest 文件经过修改满足最小特权原则以外,其他部分都没有经过修改,因此不会影响应用程序原有的结构、功能和语义。APK 合并模块使用 JDK 中的 jar 命令对 manifest 文件、dex 文件以及其他资源文件进行打包。Android 要求每个应用程序的 APK 都需要经过开发者的私钥签名,这一签名过程是不可逆的,即无法从 APK 文件中提取出原始开发者的私钥信息。而 APK 分解模块已经将原始 APK 文件中的签名破坏了,在重新合成时无法用原始开发者的私钥进行签名。本文仅采用简单随机生成的私钥使用 JDK 中的 jarsigner 工具对 APK 文件进行签名,因为 PTailor 在实际使用中可以用于 APK 签名并发布之前,帮助程序员确认其所开发的应用程序是否满足最小特权原则。而且 APK 的签名主要用于识别不同的开发者,重新合并时的签名可以针对不同的开发者使用不同的随机私钥,并不影响签名的作用。

5 实验设计与结果分析

PTailor 系统主要使用 Java 和 Python 语言实现,其中,除 API 提取模块中的 Intent 和 Content Provider 子提取模块和 APK 合并模块是使用 Python 语言实现外,其余模块都使用 Java 实现。其中 API 函数调用提取模块使用到了开源项目 dexlib2[25] 对 dex 文件进行遍历,manifest 裁剪模块使用开源项目 axml[26] 对二进制的 AndroidManifest.xml 文件进行修改。

本文通过实验对以下几个方面进行了评测和分析:PTailor 系统的性能评测、裁剪后应用程序可用性评测、应用程序权限过度声明分析、API 权限映射数据源有效性分析。

5.1 实验环境

实验中,我们对现实中的 Android 应用程序 APK 文件使用 PTailor 进行权限分析和裁剪,并进行相应的评测。实验数据集为 Google Play[27] 应用商店 27 个应用类别中销量前 50 的免费应用,去除掉重复的应用,共 1 246 个 APK 文件。实验主机内存为 16 GB,CPU 为 8 核 Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz。PTailor 系统的裁剪过程直接在实验主机上完成。裁剪后应用程序可用性评测在实验主机上的一个 Android 4.1.1 模拟器中完成。

5.2 性能评测

我们对所有1 246个 APK 文件分别使用 PTailor 进行权限分析和裁剪，并计算 PTailor 对每个 APK 文件进行分析和修改的整体时间，以及每个模块所占用的时间，从而评测 PTailor 的性能。由于 API 权限映射表生成模块是 PTailor 系统中的准备工作，不论 APK 文件数量多少，都只运行一次，所以在性能评测实验中分析 PTailor 对 APK 文件的处理时间不考虑 API 权限映射表生成模块的运行时间。

PTailor 可以成功地对所有 1 246 个 APK 文件进行分析和裁剪，成功率为 100%。所有 APK 文件的处理时间分布如图 6 和图 7 所示。图 6 为处理时间分布柱状图，即横坐标为 m 的数据柱值 n 表示有 n 个 APK 文件可以在 $m-1$ 到 m s 内被 PTailor 处理完成。图 7 为处理时间累积百分比折线图，即横坐标为 m 的数据点的纵坐标表示在 m s 内能完成 PTailor 处理的 APK 个数百分比。从图 6 中可以看出，所有的 1 246 个 APK 文件都能够在 20 s 内完成处理，而多数的 APK 文件的处理时间集中在 2～7 s 内。从图 7 中可以看出，有91.6% 的 APK 文件可以在 10 s 内完成处理。所以，PTailor 可以在很短的时间内对 Android 应用程序进行权限分析和裁剪，适用于在 Google Play 这种大量应用程序集中的 Android 应用市场中对应用程序进行自动处理。

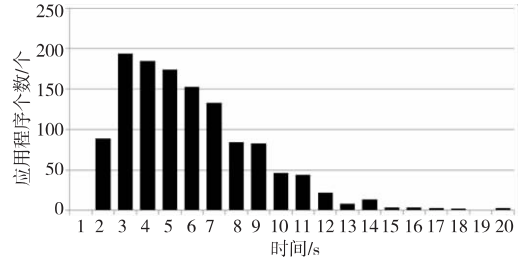

Figure 6 Processing time of PTailor

图 6 APK 文件总处理时间分布柱状图

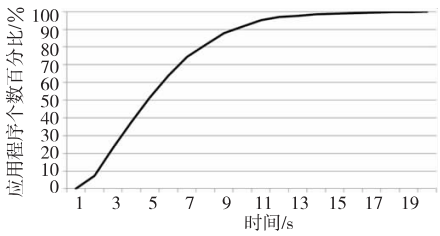

Figure 7 Cumulative percentage line graph of processing time

图 7 APK 文件总处理时间累积百分比折线图

除了对每个 APK 的整体处理时间进行统计，实验还对每个 APK 在各个模块中的处理时间进行了统计。除了 API 权限映射表生成模块外，其他不同模块的 APK 文件处理时间分布情况如表 1～表 4 所示，其中不包括 manifest 裁剪模块，因为很多应用程序不存在过度声明的权限，所以不需要进行裁剪，且大多数需要裁剪的 APK 文件的裁剪时间都在 1～2 ms 之内，由于时间过短，所以不做统计。

Table 1 Processing time of APK decomposing module

表 1 APK 分解模块处理时间分布表

时间/ms	APK 个数
≤50	268
50～100	384
100～150	208
150～200	100
>200～300	116
>300	170

Table 2 Processing time of API function call extracting module

表 2 API 函数调用提取模块处理时间分布表

时间/ms	APK 个数
≤50	462
50～100	307
100～150	189
150～200	140
>200	148

Table 3 Processing time of Intent and content provider extracting module

表 3 Intent 和 Content Provider 提取模块时间分布表

时间/s	APK 个数
≤1	3
1～2	340
2～3	380
3～4	290
>4	233

Table 4 Processing time of APK composing module

表 4 APK 合并模块处理时间分布表

时间/s	APK 个数
≤1	323
1～2	370
2～3	267
3～4	150
>4	136

从表 1 和表 2 中我们可以看出，使用 Java 实现的 APK 分解模块和 API 函数调用提取模块的

处理时间通常都在几百毫秒之内,而使用 Python 实现的 Intent 和 Content Provider 提取模块和 APK 合并模块通常都需要几秒的时间。造成这种时间差异的原因主要是使用 Python 实现的这几个模块都通过调用外部命令的方式进行分析和处理,例如 find、grep、jar、jarsigner 等等,并且这些命令都要频繁进行文件的 IO 操作,调用外部命令和 IO 操作通常都需要花费较多时间。

5.3 可用性评测

可用性评测的目的是检验 PTailor 对 APK 的裁剪是否过度,经过裁剪的应用是否依然能够正常运行。实验采用 Android SDK 中的自动化压力测试工具 Monkey。

Monkey 通过将随机生成的 UI 事件序列注入到应用程序中的方式对应用程序进行压力测试。在 Monkey 中可以设置事件序列中的事件个数,并可以指定只针对某一个应用进行测试。同时,可以通过设置随机种子的方式,指定 UI 事件序列的顺序。Monkey 在程序运行结束后给出详细的测试报告。如果应用程序在某一个 UI 事件后崩溃,Monkey 将停止测试并自动退出,并报告该程序已经崩溃。另外 Monkey 还有可能因为无法找到应用程序的入口等原因而意外退出。使用 Monkey 进行压力测试的目的是检查 PTailor 的裁剪是否会影响程序的可用性。

实验中,我们首先对所有未裁剪的 APK 文件进行 Monkey 测试,每个测试的随机事件个数为 2 000,并记录下对每个 APK 进行测试时所使用的随机种子 seed。同时,我们还会记录下未裁剪的应用中崩溃的应用以及导致 Monkey 测试失败的应用,这些未裁剪就已经崩溃和意外退出的应用是由其本身错误或者与模拟器版本不兼容所导致的,对实验没有意义,所以在之后对裁剪过的 APK 文件进行 Monkey 测试时将不考虑这些已崩溃的应用。

然后,我们对裁剪过的 APK 文件进行 Monkey 测试。每个裁剪过的 APK 文件进行测试时所使用的随机种子与其对应的未裁剪版本在进行测试时所记录下的随机种子 seed 相同,这使得裁剪前和裁剪后的同一个 APK 文件在进行 Monkey 测试时使用相同的 UI 事件序列。同样地,我们也记录下裁剪过的应用程序中崩溃的以及导致 Monkey 意外退出的程序。这些应用的崩溃可能是由于 PTailor 裁剪了过多权限所导致的,需要进行进一步分析。

在实验过程中,对每一个 APK 文件的测试都单独进行,每一次测试只安装一个 APK 文件,对其进行 Monkey 测试结束后,将该 APK 文件卸载,从而避免对实验环境的改变以及对其他应用测试的影响。但是,在安装 APK 文件时,也会有一些 APK 文件安装失败,因此我们也会记录安装失败的 APK 文件,并只对安装成功的 APK 文件进行测试。整个实验过程如图 8 所示。

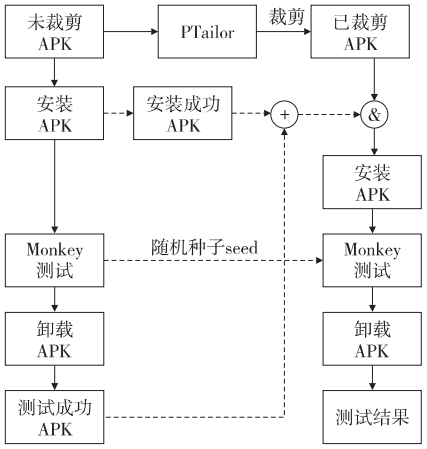

Figure 8 Workflow of usability experiment

图 8 可用性测试实验流程

在可用性测试结束后,对可用性测试结果进行统计和总结。在对未裁剪的 APK 文件进行 Monkey 测试时,共有 73 个 APK 文件安装失败,有 202 个 APK 文件崩溃或导致 Monkey 测试失败。排除这些无意义的 APK 文件外,共有 971 个应用程序在未裁剪时能成功运行。对这 971 个应用程序的裁剪后版本进行 Monkey 测试,共有 69 个应用程序崩溃,而这 69 个应用程序中,有 17 个应用程序在 PTailor 分析结果中并没有权限过度声明现象,因此,这 17 个应用程序实际并没有被裁剪,导致其崩溃的原因可能是网络环境影响等因素,与 PTailor 权限裁剪无关。而剩余的 52 个崩溃的应用程序,只占全部 1 246 个应用程序的 4.17%,占全部可运行的 971 个应用程序的 5.35%,从而可以说明 PTailor 的权限裁剪只会影响很少的 Android 应用程序,大多数被裁剪的程序依然能够正确运行。

5.4 应用程序权限过度声明分析

使用 PTailor 对全部 1 246 个 Android 应用程序进行权限裁剪,我们发现一共有 811 个应用存在权限过度声明的现象,占全部应用程序数量的 65.1%。Felt A P 等人[2]在 2011 年使用 Stowaway

对采集到的 940 个应用进行了分析，发现 34.4% 的应用存在权限过度声明现象。Au K W Y 等人[4]在 2012 年使用 PScout 对采集到的 1 260 个应用进行了分析，发现 43.1% 的应用存在权限过度声明现象。将 PTailor 与这两个工作相比较，从图 9 中可以看出，从 2011 年到 2014 年，Android 应用程序权限过度声明现象呈现上升趋势，这与文献[5]中分析不同版本应用程序得出权限过度声明程序数量随时间不断增加的结论基本一致。

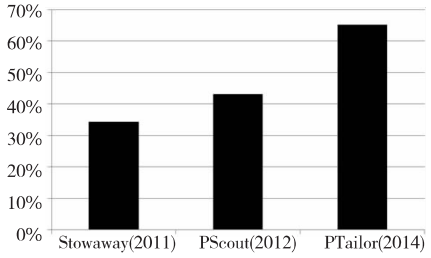

Figure 9 Comparison of overprivileged applications

图 9 权限过度声明比例比较

在这 811 个权限过度声明应用程序中，30.57% 的应用只过度声明了一个权限，49.25% 的应用过度声明权限个数小于或等于 2 个，74.88% 的应用过度声明权限个数小于或等于 4 个。具有不同过度声明权限个数的应用程序数量分布柱状图如图 10 所示。

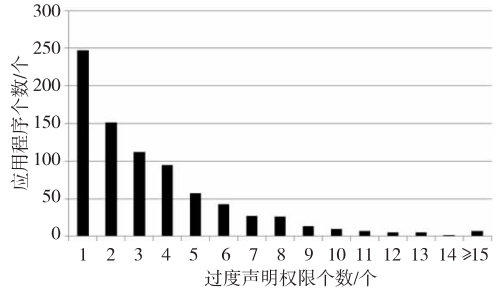

Figure 10 Distribution of overprivileged permissions

图 10 应用程序过度声明权限个数分布柱状图

另外我们发现，在所有 1 246 个应用程序中，有很多应用声明了名称以 android.permission. 和 com.android. 为开头但在 Android 系统中并不存在的权限。通常，名称以 android.permission. 和 com.android. 为开头的权限是 Android 的系统权限。这些以 android.permission. 和 com.android. 为开头但在 Android 系统中并不存在的权限可能是某些 Android 应用程序自定义的权限，但是也有一些是根本不存在甚至是明显书写错误的权限。这些不存在或书写错误的权限是由于程序开发者的个人因素所导致的，应该尽量避免。例如，

(1)与系统权限名称相近，但根本不存在的权限。程序如果希望读取短信，则需要声明 READ_SMS 权限，但有些应用程序却声明了 READ_MMS 这个根本不存在的权限。

(2)无需声明且根本不存在的权限。Android 为每个应用程序提供独立私有的内部存储空间用于存储数据，使用这个内部空间并不需要声明任何权限，但有些应用程序却声明了 READ_INTERNAL_STORAGE 这个根本不存在的权限。

(3)书写错误的权限。应用程序如果要录制音频，则需要声明 RECORD_AUDIO 权限，但有些应用程序开发者却将这一权限错误地书写成了 RECORDE_AUDIO。

5.5 API 权限映射数据源有效性分析

4.1.1 节中提到，PTailor 现阶段的 API 权限映射数据源来自于 PScout[4] 的实验结果，据我们所知，PScout 中所得到的 API 权限映射关系是目前最为完整的，其对 Android 4.1.1 的分析结果中包括了 32 450 个 API 函数直接调用权限映射，97 个 Intent Action 权限映射和 78 个 Content Provider URL schema 权限映射，但经过实验我们发现该数据源并不完善，存在很多缺陷。

5.5.1 权限覆盖率分析

PScout 对 Android 4.1.1 中的 API 权限映射进行分析，共找到了 101 个权限与 API 有所映射，但 Android 4.1.1 系统中的所有权限个数为 180 个。其中，Android 系统中有 92 个权限在 PScout 中没有得到对应，而在全部 1 246 个应用中有 137 个程序所声明的权限包含在这 92 个 PScout 未对应的权限中。另外，PScout 的 API 权限映射结果中，有 13 个权限是 Android 系统提供给系统级应用程序的特殊权限，无法被普通应用所使用。从以上结果中可以看出，PScout 的权限覆盖率仅为 48.9%。

5.5.2 API 映射分析

除了权限覆盖不完整之外，PScout 中的 API 权限映射关系也不完整。具体包括以下几点：

(1)外部存储设备访问权限。如果应用程序想读取或写入 SD 卡中的文件，访问 sdcard 目录，则该应用必须具有 READ_EXTERNAL_STORAGE 或 WRITE_EXTERNAL_STORAGE 权限，同时为了对 SD 卡上的文件进行操作，有些应用程序还需要 MOUNT_UNMOUNT_FILESYSTEMS 权限。但是，PScout 的 API 权限映射中没有这些与 SD 卡文件操作相关的权限映射。

(2)WebView 访问网络权限。Android 系统提供 WebView 组件用于显示 Web 页面,任何使用 WebView 显示 Web 页面的应用程序都应该具有 INTERNET 权限。WebView 可以在程序的代码段中动态声明,也可以在 xml 文件中静态声明。PScout 的 API 权限映射结果中只包含 WebView 动态声明 API,而没有 xml 静态声明 WebView 的权限映射。

(3)读取 Log 权限。Android 系统向应用提供用于记录信息的 Log 接口,任何使用 logcat 命令读取 Log 的应用程序都需要具有 READ_LOGS 权限,该权限信息在 PScout 中没有映射。

(4)漏报的 API。PScout 中虽然含有超过 30 000个 API 与其权限的映射,但仍有一些需要权限的 API 没有得到映射。例如,android.hardware.Camera 中的 open(int)函数需要 CAMERA 权限,但 PScout 中没有该 API 的权限映射。

(5)不同参数对应的权限。对于有些 API 函数,当使用不同参数进行函数调用时,其所需的权限可能不同。例如,当应用程序打开一个具有 TYPE_SYSTEM_ALERT 参数的窗口,使窗口出现在所有其他窗口上时,程序应该具有 SYSTEM_ALERT_WINDOW 权限。但是,PScout 没有函数调用参数的权限映射信息。

6 讨论

6.1 应用程序之间的请求

Android 允许一个应用程序请求另一个应用程序进行某个操作、完成某项任务,这种请求只能通过消息传递的方式进行通知,而无法进行直接的函数调用。由于程序之间不能够直接调用函数,而且程序之间也不存在请求传递的关系,如果被请求方将请求方的系统资源请求直接转发出去,则请求方和被请求方都应该具有相应的权限。因此,并不存在某一个应用程序为其他应用程序预先申请权限、使得无权限一方调用有权限一方的情况,因此 PTailor 的裁剪不会影响程序之间的请求。

6.2 可用性评测方法局限性

5.3 节中使用 Monkey 对 PTailor 裁剪后应用程序的可用性进行了评测。Monkey 虽然可以通过随机注入事件序列的方式对应用程序进行自动化的压力测试,但这种方法具有一定的局限性。由于 Monkey 的事件类型和注入顺序都是随机的,所以使用 Monkey 测试并不能保证完全的代码覆盖率,可能导致测试的不完整,增加漏报率。但是,现阶段还没有代码覆盖率为 100% 的 Android 应用程序自动化测试方法,因此,本实验的漏报率还无法测量。在未来的工作中,我们将设计并提出代码覆盖率更高、更符合程序流程结构的 Android 应用程序自动化测试方法。

6.3 导致裁剪后应用程序不可用的原因

从 5.3 节的可用性评测结果中可以看出,PTailor 的权限裁剪依然会影响很少一部分 Android 应用程序的正常运行。导致这一结果的主要原因有以下几点:

(1)API 权限映射数据源的不完整性。PTailor 可以使用任何 API 权限映射信息作为其数据源,本文中使用 PScout[4] 中得到的 Android 4.1.1 API 权限对应结果作为 API 权限映射数据源。据我们所知,该数据源中的 API 权限映射关系是目前最为完整的。但是,在 5.5 节中的 API 权限映射数据源有效性分析中可以看出该数据源的 API 权限映射关系并不完整,这可能导致 PTailor 将一些实际需要权限、但在数据源中没有得到映射的 API 裁剪掉,从而导致过度裁剪,影响程序的正常运行。在未来的工作中,我们计划研究并提出能够发现更多 API 权限映射关系的方法,从而提高 API 权限映射数据源的完整性,减少 PTailor 的过度裁剪。

(2)Java 反射机制的影响。Android 应用程序使用 Java 语言编写,Java 提供反射机制允许应用程序在运行时动态地调用函数。PTailor 的 API 函数调用提取模块中没有对反射机制的函数调用进行分析,完整的反射机制函数调用分析需要进行信息流敏感、跨函数的静态分析。在未来的工作中,我们将在 PTailor 中增加对反射机制函数调用的静态分析。

这些因素有可能引起 PTailor 由于无法找到某些权限在程序中所对应的 API 而导致的权限过度裁剪,但是不会导致应用程序过度声明的权限没有得到裁剪。即经过 PTailor 裁剪的应用程序权限集合,可能是其最小特权的子集,但不会是最小特权的超集。但是,由于现阶段没有完整的 API 权限映射数据,所以暂时无法证明经过 PTailor 裁剪的权限集合是否与应用程序的最小特权集合完全吻合。

7 结束语

Android 应用程序的权限声明应该满足最小特权原则，但现实中的很多应用程序声明了过多不使用的权限，从而可能带来安全隐患。本文提出了一种 Android 应用程序权限自动裁剪系统 PTailor，PTailor 在应用程序中提取系统 API 调用信息，分析调用这些 API 所需的对应权限，得到应用程序实际所需要的最少权限，并将程序中多余的未被使用的权限裁剪掉，从而使得裁剪后的 Android 应用程序满足最小特权原则。使用 PTailor 对现实中的 1 246 个 Android 应用程序进行权限分析和裁剪实验，实验结果表明 91.6%的 APK 文件能够在 10 秒内完成权限分析和裁剪，最长的处理时间为 20 秒，并且 PTailor 的裁剪不会对大多数程序的运行产生明显影响。同时，文中还对应用程序权限过度声明趋势进行了分析，发现权限过度声明的应用程序数量随时间呈现上升趋势，且应用程序存在因开发者个人因素所导致的错误权限声明现象。另外，通过实验我们还发现，以往的 API 权限映射分析研究工作还存在很多不足和缺陷。

参考文献：

[1] Android captured 79% share of global smartphone shipments in 2013 [EB/OL]. [2014-05-16]. http://blogs.strategyanalytics.com/WSS/post/2014/01/29/Android-Captured-79-Share-of-Global-Smartphone-Shipments-in-2013.aspx.

[2] Felt A P, Chin E, Hanna S, et al. Android permissions demystified[C]// Proc of the 18th ACM Conference on Computer and Communications Security, 2011:627-638.

[3] Felt A P, Egelman S, Finifter M. et al. How to ask for permission[C]// Proc of HotSec'12, 2012:1.

[4] Au K W Y, Zhou Y F, Huang Z, et al. Pscout: Analyzing the Android permission specification[C]// Proc of the 2012 ACM Conference on Computer and Communications Security, 2012:217-228.

[5] Bartel A, Klein J, Le Traon Y, et al. Automatically securing permission-based software by reducing the attack surface: An application to Android[C]// Proc of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012:274-277.

[6] Vidas T, Christin N, Cranor L. Curbing Android permission creep[C]// Proc of the Web 2.0 Security and Privacy 2011, 2011:1.

[7] Wei X, Gomez L, Neamtiu I, et al. Permission evolution in the Android ecosystem[C]// Proc of the 28th Annual Computer Security Applications Conference, 2012:31-40.

[8] Yang Bo, Tang Zhu-shou, Zhu Hao-jin, et al. Method of Android applications permission detection based on static dataflow analysis[J]. Computer Science, 2012, 39(11A):16-18. (in Chinese)

[9] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification[C]// Proc of the 16th ACM Conference on Computer and Communications Security, 2009:235-245.

[10] Barrera D, Kayacik H G, van Oorschot P C, et al. A methodology for empirical analysis of permission-based security models and its application to Android[C]// Proc of the 17th ACM conference on Computer and Communications Security, 2010:73-84.

[11] Peng H, Gates C, Sarma B, et al. Using probabilistic generative models for ranking risks of Android apps[C]// Proc of the 2012 ACM Conference on Computer and Communications Security, 2012:241-252.

[12] Pearce P, Felt A P, Nunez G, et al. AdDroid: Privilege separation for applications and advertisers in Android[C]// Proc of the 7th ACM Symposium on Information, Computer and Communications Security, 2012:71-72.

[13] Zhang Y, Yang M, Xu B, et al. Vetting undesirable behaviors in Android apps with permission use analysis[C]// Proc of the 2013 ACM SIGSAC Conference on Computer & Communications Security, 2013:611-622.

[14] Yang Huan, Zhang Yu-qing, Hu Yu-pu, et al. A malware behavior detection system of Android applications based on multi-class features[J]. Chinese Journal of Computers, 2014, 37(1):15-27. (in Chinese)

[15] Zhang Rui, Yang Ji-yun. Android malware detection based on permission correlation[J]. Journal of Computer Applications, 2014, 34(5):1322-1325. (in Chinese)

[16] Nauman M, Khan S, Zhang X. Apex: Extending Android permission model and enforcement with user-defined runtime constraints[C]// Proc of the 5th ACM Symposium on Information, Computer and Communications Security, 2010:328-332.

[17] Bao Ke-jin, Peng Zhao. An extended Android application permission management model[J]. Computer Engineering, 2012, 38(18):57-60. (in Chinese)

[18] Xu R, Saïdi H, Anderson R. Aurasium: Practical policy enforcement for Android applications[C]// Proc of the 21st USENIX Conference on Security Symposium, 2012:27.

[19] Livshits B, Jung J. Automatic mediation of privacy-sensitive resource access in smartphone applications[C]// Proc of the 22nd USENIX Security Symposium, 2013:113-130.

[20] Smalley S, Craig R. Security enhanced (se) Android: Bringing flexible MAC to Android[C]// Proc of the 20th Annual Network and Distributed System Security Symposium (NDSS'13), 2013:1.

[21] Bugiel S, Heuser S, Sadeghi A R. Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies[C]// Proc of the 22nd USENIX Security Symposium (USENIX Security'13), 2013:131-146.

[22] Bugiel S, Davi L, Dmitrienko A, et al. Practical and lightweight domain isolation on Android[C]// Proc of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, 2011:51-62.

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

[24] Tang Wei. Research and improvement on Android framwork's security enforcement [D]. Ningbo: Ningbo University, 2011. (in Chinese)

[25] Smali [EB/OL]. [2014-05-16]. https://code.google.com/p/smali/.

[26] axml [EB/OL]. [2014-05-16]. https://code.google.com/p/axml/.

[27] Google Play[EB/OL]. [2014-05-16]. https://play.google.com/store.

附中文参考文献:

[8] 杨博, 唐祝寿, 朱浩谨, 等. 基于静态数据流分析的 Android 应用权限检测方法[J]. 计算机科学, 2012, 39(11A):16-18.

[14] 杨欢, 张玉清, 胡予濮, 等. 基于多类特征的 Android 应用恶意行为检测系统[J]. 计算机学报, 2014, 37(1):15-27.

[15] 张锐, 杨吉云. 基于权限相关性的 Android 恶意软件检测[J]. 计算机应用, 2014, 34(5):1322-1325.

[17] 鲍可进, 彭钊. 一种扩展的 Android 应用权限管理模型[J]. 计算机工程, 2012, 38(18):57-60.

[24] 汤伟. Android 应用程序框架安全机制研究及改进[D]. 宁波:宁波大学, 2011.

作者简介:

白小龙(1989-),男,吉林集安人,博士生,CCF 会员(E200037503G),研究方向为手机安全。**E-mail**:baixiaol@sina.com

BAI Xiao-long, born in 1989, PhD candidate, CCF member (E200037503G), his research interest includes mobile security.