

基于静态分析的 Android 应用事件输入生成方法

叶益林, 周振吉, 洪征, 颜慧颖, 吴礼发

(解放军理工大学指挥信息系统学院, 江苏 南京 210007)

摘要: 提出了一种基于静态分析的 Android 应用动态分析事件输入生成方法。以组件间调用图和单个组件内部系统依赖图为基础, 提取组件内部安全相关回调方法依赖的事件输入, 并设计了一种事件输入算法, 自动生成 Android 应用运行时依赖的事件输入。实验结果表明, 相比已有工作, 所提方法能够达到更高的权限方法和基本组件覆盖率, 覆盖更多安全相关的执行路径, 有助于动态分析过程中收集更多安全相关的运行时行为。

关键词: Android; 事件输入; 动态分析; 系统依赖图

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2017.00169

Static-analysis-based event input generation approach for Android application

YE Yi-lin, ZHOU Zhen-ji, HONG Zheng, YAN Hui-ying, WU Li-fa

(Institute of Command Information System, PLA University of Science and Technology, Nanjing 210007, China)

Abstract: A static-analysis-based event input generation approach for Android applications was proposed. Based on the inter-component call graph and the system dependence graph of single component, the event inputs that security-related callbacks depend on were extracted. Furthermore, an event input algorithm was designed to automatically generate event inputs according to Android application runtime. The experimental results show that the proposed method can achieve higher coverage of permission methods and basic components, which can cover more security-related execution paths and help to collect more security-related runtime behaviors during the process of dynamic analysis.

Key words: Android, event input, dynamic analysis, system dependence graph

1 引言

动态分析技术通过执行应用代码, 收集并根据应用程序的动态行为判别应用的安全性, 能够克服静态分析容易被动态加载、反射等技术手段绕过的问题。一个完备的 Android 应用动态分析方案包含 3 个部分: 输入生成、行为收集和行为分析。输入生成是指生成触发 Android 应用程序

安全相关行为输入事件的过程; 行为收集是指在动态分析过程中收集 Android 应用程序运行时行为的过程; 行为分析是指对 Android 应用程序运行时行为进行分析的过程。

Android 应用是一类交互式应用, 依赖外部事件触发不同的功能逻辑, 不同的输入会导致不同的执行路径。输入事件决定动态分析时会执行的路径, 从而决定运行时收集到的行为, 是动态

收稿日期: 2017-04-13; 修回日期: 2017-05-22。通信作者: 叶益林, my_etsi@163.com

基金项目: 江苏省自然科学基金资助项目 (No.BK20131069)

Foundation Item: The Natural Science Foundation of Jiangsu Province (No.BK20131069)

分析有效与否的前提。已有 Android 恶意应用动态分析方案^[1~5]侧重于解决如何收集 Android 应用动态运行时行为的问题, 缺乏对如何生成输入事件、触发安全相关路径的研究。这些工作通常基于 Monkey^[6]或 MonkeyRunner^[7]生成动态分析所需的各类输入事件。Monkey 是一个类似 Fuzzing 的工具, 能够生成无序伪随机事件, 常用于 Android 应用压力测试。MonkeyRunner 是 Android 官方提供的另一类测试工具, 提供了用户编程接口, 支持基于 Python 脚本生成各类定制的事件。由于缺乏对 Android 应用具体功能逻辑的分析, 尤其缺乏如何触发安全相关执行路径的分析, 上述 2 种方法生成的输入事件都缺乏针对性, 难以触发安全相关路径。

动态分析 Android 恶意应用的关键在于尽可能多地触发安全相关的执行路径。Android 应用恶意与否通常只取决于安全相关代码(安全相关代码是指那些会调用敏感 API 执行路径上的应用代码), 故 Android 动态分析的目标是尽可能多地执行安全相关代码。然而, 已有研究通常以代码覆盖率作为关键性能指标, 这种做法难以保证安全相关代码的覆盖率, 因而难以触发更多的安全相关行为。与此同时, 由于异常代码、不可达代码以及需要其他特殊条件才能触发代码的存在, 动态分析通常难以实现高代码覆盖率。如何在保持一定代码覆盖率的情况下, 尽可能地提高安全相关代码的覆盖率, 从而触发尽可能多的 Android 应用程序运行时安全相关行为是本文重点解决的问题。

本文提出了一种基于静态分析的 Android 应用动态分析输入事件生成方法。首先, 对样本进行静态分析, 确定样本中全部 UI 事件回调方法和系统事件回调方法; 然后, 对样本构建组件间调用图, 对各个基本组件构建系统依赖图(类依赖图、类层次图、过程依赖图、控制依赖图、数据依赖图); 基于组件间调用图和单个组件对应的系统事件图, 利用程序切片方法, 提取触发两类回调方法所需的输入事件; 最后, 设计事件输入算法完成输入事件过程。为衡量本文方法的有效性, 首次提出以权限方法覆盖率、基本组件覆盖率作为主要性能指标。本文的主要创新及贡

献有: 1) 提出一种基于静态分析的 Android 恶意应用动态分析事件输入生成方法; 2) 基于 Android 应用程序的组件间调用图和组件依赖图, 自动提取安全相关回调方法的事件依赖, 并设计一种 Android 应用事件序列输入算法。

2 方法描述

Android 应用是基于事件驱动、交互式的应用程序, 执行流程可以简单归结为在各类输入事件的驱动下, 应用程序完成各种不同的逻辑功能。Android 应用程序处理两类输入事件: UI 事件和系统事件。Android 系统通过 UI 事件回调方法和系统事件回调方法响应上述 2 种输入事件。由于 Android 权限系统的限制, 所有能够访问系统敏感资源或改变敏感系统配置的 API, 均需要相应的权限才能被调用。通常这些敏感权限 API 在 UI 事件回调方法和系统事件回调方法中被调用。本文将那些调用了敏感权限 API 的回调方法称为安全相关回调方法, 因而动态分析 Android 恶意应用的关键在于如何触发那些包含敏感权限 API 的安全相关回调方法。对于那些不在回调方法内部调用的敏感 API, 如果不依赖外部输入, 通常只需遍历这些敏感 API 所属的基本组件便能触发。若依赖于外部输入, 则需要路径求解获取依赖条件, 从而触发这些敏感 API。

综上, 动态分析 Android 应用的基本前提是解决如何生成能够触发安全相关回调方法所需的输入事件, 具体可以归结为 3 个问题: 输入什么事件、如何生成所需事件以及如何输入所需事件。下面从事件提取、事件生成、事件输入这 3 个方面阐述如何解决上述 3 个问题。

2.1 事件提取

Android 应用程序内部回调方法必须与某个 UI 事件或系统事件关联, 而且必须附属于某个基本组件。本文方法将 Android 应用程序看成一个有限状态机, 每个基本组件对应一个程序状态。每个状态下, 应用程序都能响应 UI 事件或系统事件, 使 Android 应用从一个状态切换到另一个状态。一般地, 在某个状态下, 触发一个回调方法所需的输入事件可以分为 2 个部分, 用二元组 (Pre, Con) 表示。其中, Pre 表示到达当前状态

所需的输入事件, Con 表示当前状态下触发回调方法所需的输入事件。

Pre 与触发 Android 应用组件间跳转的事件相关, 可以通过构造 Android 应用的全局基本组件间调用图求解。基本组件间调用图的节点为 Android 应用程序中所有基本组件, 边表示组件间调用关系以及调用条件, 其中, 调用条件是指从当前组件转移到下一个组件所依赖的外部输入。为确定 Con, 可以构造单个基本组件的系统依赖图, 并以目标回调方法调用点作为切片标准, 对系统依赖图进行程序切片, 提取触发目标回调方法所需的语句或谓词, 再通过分析这些语句和谓词确定触发目标回调方法所需的事件, 包括以下内容: 事件类型、事件的个数及其先后次序, 以及所需的数据输入。

2.1.1 组件间调用图

Android 应用包括 Activity、Service、Broadcast Receiver 和 Content Provider 四大组件, 组件间调用图描述了 Android 应用内部所有组件间的跳转关系。构造该图需要了解 Android 基本组件常见的跳转方法, 组件间跳转借助 Intent 完成, 在 Intent 内部封装包含目的组件在内的各种数据, 常用的组件间跳转方法如表 1 所示。

本文采用算法 1 构造组件间调用图, 首先利用 apktool 解析样本 apk, 将源码转换成 smali 文件, 并提取 AndroidManifest.xml 文件。分析 AndroidManifest.xml 文件, 提取样本中所有静态注册的基本组件。根据组件名, 分析组件对应的 smali 源码文件, 判断是否调用了表 1 中的跳转方法。若调用了其中的方法, 则分析调用跳转方法的上下文, 确定目的组件, 并在组件间调

用图中添加一条从当前组件到目的组件的边。此外, Android 系统只支持动态创建 Broadcast Receiver, 因而需要在分析源码时 (算法 1 第 7 行), 判断组件内部是否存在动态创建的广播接收者。对于当前组件内部动态创建的组件, 直接在组件间调用图添加一条从当前组件到动态创建组件的边 (算法 1 第 9、10 行)。

算法 1 组件间调用图生成

输入 apk

输出 CCG, Component Call Graph

```

1) CCG=(V, E)
2) smali_files, AndroidManifest.xml ←
   parse(apk)
3) components ← {analyze AndroidManifest.
   xml and extract all components defined in it}
4) for com in components do
5)   V ← {add com to V}
6)   source_code ← search(smali_files, com)
7)   transfer_methods, dyn_coms ← Analyze
   (source_code)
8)   for dyn in dyn_coms do
9)     V ← {add com to dyn}
10)    E ← {add an edge from com to dyn}
11)   done
12)   for meth in transfer_methods do
13)     target ← {extract the targeted com-
   ponent by analyzing the context where meth is
   called}
14)     V ← {add target to V}
15)     E ← {add an edge from com to tar-
   get}

```

表 1 Android 系统常用组件间跳转方法

方法	描述
startActivity	启动一个 Activity, 由参数 Intent 指定
startActivityForResult	启动一个 Activity, 由参数 Intent 指定。新 Activity 结束后, 返回
sendBroadcast	发送一个广播, 广播接收者由 Intent 内封装的数据指定
sendOrderBroadcast	发送一个按序事件广播, 具有权限的接收者按优先级高低接收广播
sendStickyBroadcast	发送一个留存广播, 接收者处于活跃状态时, 便能接收到广播
startService	启动一个服务, 服务对象由 Intent 指定
bindService	绑定一个服务, 服务对象由 Intent 指定

- 16) done
- 17) done
- 18) Output CCG

2.1.2 组件内部系统依赖图

得到组件间调用图后，只是确定了组件间的调用关系链，还需要构建单个组件的系统依赖图，通过分析单个组件的系统依赖图，才能确定 Pre 和 Con,从而构造触发回调方法的输入事件链。本文方法直接使用文献[8, 9]中的方法构造单个组件的系统依赖图。Android 应用程序单个组件的系统依赖图由过程依赖子图、类依赖子图、类层次子图、控制依赖子图、数据依赖子图组成。

由于 Android 应用程序的特殊性，在构造单个组件的系统依赖图时，需要注意隐式调用问题。隐式调用是指 Android 应用源码中 2 个无显式调用关系的方法，在 Framework 层代码的衔接下存在调用关系的现象。Android 系统中有 3 种情况存在隐式调用：一是 Android 应用响应外部事件的回调方法；二是异步调用方法；三是生命周期方法。

1) 回调方法

在 Android 应用中，回调方法注册点与调用点通常处于分离状态，两者没有显式的调用关系。在分析源码时，需要根据 Android 支持的回调方法注册样式，确定注册点和调用点。Android 系统 UI 事件回调方法和系统事件回调方法常见的

注册方式如表 2 所示。

表 2 Android 系统回调方法注册方式	
回调方法	注册方式
UI 事件回调方法	匿名内部类实现
	内部类实现
	接口实现
	属性方法
系统事件回调方法	系统事件监听器
	系统事件广播接收者
	系统内容提供者回调方法

Android 系统中，所有 UI 都是 View 对象的子类。UI 事件回调方法注册点为事件监听器(event listener)，注册事件监听器有多种方式。UI 事件回调方法通常由对应的事件监听器触发调用，具体如表 3 所示。

系统事件监听器是 Android 系统以接口的形式向用户提供的一种系统事件监听方式，典型的系统事件监听器有地理位置变更监听器(location state listener)和手机状态监听器(phone state listener)。系统事件广播者是注册事件回调方法的主要方式。系统事件发生后，Framework 发出系统广播，注册了相关系统事件的 Android 应用程序将被激活，Framework 将调用该应用程序广播接收者的 onReceive 方法。Android 系统提供了工具类内容观察者(content observer)用于捕获系统内容提供者中的数据状态的改变事件。当指定

表 3 View 对象支持的事件监听器及其回调方法			
事件监听器	注册方式	回调方法	事件类型
OnAttachStateChangeListener	setOnAttachStateChangeListener	onAttachStateChange	关联状态改变事件
OnClickListener	setOnClickListener	onClick	单击事件
OnCreateContextMenuListener	setOnCreateContextMenuListener	onCreateContextMenu	菜单创建事件
OnDragListner	setOnDragListener	startDrag	拖拽事件
OnFocusChangeListener	setOnFocusChangeListener	onFocusChange	焦点改变
OnGenericMotionListener	setOnGenericMotionListener	onGenericMotionEvent	普通移动事件
OnHoverListener	setOnHoverListener	onHover	悬停事件
OnKeyListener	setOnKeyListener	onKey	键盘事件
OnLayoutChangeListener	setOnLayoutChangeListener	onLayoutChange	布局改变
OnLongClickListner	setOnLongClickListener	onLongClick	长按事件
OnTouchListener	setOnTouchListener	onTouch	触摸事件

内容提供者的内容发生改变后, Framework 调用内容观察者中的 `onChange` 方法。

在构造类依赖子图时, 需要在注册点与调用点之间添加一条控制依赖边。对于在 `AndroidManifest.xml` 中通过 UI 控件属性方法注册的回调方法, 将其 UI 控件所属 Activity 的 `onCreate` 方法作为控制依赖边的源节点。

2) 异步调用方法

Android 系统内部大量使用异步编程模式, 一个典型的应用场景为确保 UI 线程能够及时响应用户操作, 保证用户体验。Android 系统编程规范强制要求将时延长、响应慢的任务在后台运行, 实现这一机制的方式便为异步调用。Android 系统中常用的异步调用可以分为继承 Java 的异步调用和 Android 系统特有的异步调用这两类。Java 语言常用的异步调用通过 `Thread` 类实现 (或 `Runnable` 接口), 异步调用执行流程为 `Thread.start`→`Thread.run`。Android 系统特有的异步调用包括 `Handler` 和 `AsyncTask` 这 2 个工具类。`Handler` 类的异步调用执行流程为 `sendMessage`→`handleMessage`。`AsyncTask` 是 Android 系统特有的用于后台线程与 UI 线程通信的工具类, 4 个异步方法的执行流程为 `execute` (UI 线程)→`onPreExecute` (UI 线程)→`doInBackground` (后台线程)→`postExecute` (UI 线程)。在构造类依赖子图时, 在异步调用源方法与目标方法之间增加一条控制依赖边。

3) 生命周期方法

与 X86 平台应用不同, Android 应用具有严格的生命周期方法。在构造类依赖图时, 需要根据生命周期方法调用关系, 在不同生命周期方法间建立控制依赖关系。Android 应用只有处于 `Active/Running` 状态、获取用户焦点后, 才能响应 UI 事件。在执行 `onResume` 方法后, Android 程序状态切换到 `Active/Running` 状态。由此, 本文方法以 `onResume` 方法作为 UI 事件回调方法的调用点。因而在类依赖子图中, 对所有 UI 事件回调方法, 增加一条从 `onResume` 方法到 UI 事件回调方法的调用边。对于系统事件回调方法, 遵循以下原则构造类依赖子图。

① 系统广播接收者

对于静态注册的系统广播接收者, 以其唯一的生命周期方法 `onReceive` 作为系统依赖图和类依赖子图的入口点。对于动态注册的广播接收者, 需要添加一条从注册点到 `onReceive` 方法的控制依赖边, 其他同静态注册的广播。

② 系统事件监听类

与动态注册的广播接收者一样, 在注册点处方法与实际调用点方法间增加一条控制依赖边。

Android 应用没有 `Main` 方法, 但具有多个程序入口点。在构造单个组件的系统依赖图时, 需要确定其入口点。本文采用以类依赖子图的入口点作为系统依赖图的入口点的策略。由于基本组件生命周期方法具有严格的执行顺序, 本文方法以基本组件第一个执行的生命周期方法作为入口点。对于 Activity, 以 `onCreate` 方法作为入口点; 对于 Receiver, 以 `onReceive` 方法作为入口点; 对于 Service, 若应用重构了 `onCreate` 方法, 则以 `onCreate` 方法作为入口点, 否则以 `onStartCommand` 方法作为入口点。

2.2 事件生成

Android 应用程序组件的系统依赖图同时包含安全无关和安全相关的回调方法, 本文方法利用 `Pscout`^[10]提供的权限与 API 之间的映射关系, 确定应用中所有安全相关回调方法。需要说明的是, 对于那些不在回调方法中调用的敏感权限 API, 由于不需要任何事件触发, 只要在动态分析过程中, 当程序状态迁移到这些敏感权限 API 所属的基本组件时, 这些 API 将会被自动调用。

1) UI 事件回调方法

UI 事件回调方法对外部事件的依赖, 主要体现在数据依赖和控制依赖这 2 个方面。为确定 UI 事件回调方法的外部依赖关系, 需要遍历 UI 事件回调方法所关联组件对应的系统依赖图。首先, 从系统依赖图中, 定位 UI 事件回调方法 (假定为 A) 所在的过程依赖子图, 确定过程依赖子图中节点所有依赖外部数据或控制的语句或谓词。然后, 在数据依赖子图和控制依赖子图中进行后向切片, 确定 UI 事件回调方法内变量对过程外元素的依赖关系链。分析这些数据依赖和控制依赖关

系链:若依赖关系中存在对其他某个 UI 事件回调方法(假定为 B)的数据依赖或控制依赖,则说明触发 A 需要先触发 B;同理,若 B 依赖于某个回调方法 C,则说明触发 A 需要先触发 C。依此类推,分别提取各个回调方法依赖的输入事件,将这些输入事件逆序排列便是触发目标回调方法所需的事件链。

需要注意的是,A 与 B 之间存在数据依赖边,则说明用户在触发 B 时,需要输入相关数据,而 A 将使用用户输入的数据。在动态分析时,需要输入合法的数据,才能触发 A 中相关的功能逻辑。对于这种需要外部输入数据的情况,采用如下解决方法。

分析外部数据在 A 中的使用情况,判断外部输入是否会影响 A 中安全相关 API 的调用。如果安全相关 API 与外部数据不存在数据依赖和控制依赖,则说明外部输入对安全相关 API 的调用毫无影响,可以随机输入。外部输入数据影响其调用情况如下:若是数据依赖,即外部输入作为安全相关 API 的参数,则根据外部输入数据在输入时的上下文,构造相关外部输入;若是控制依赖关系,则分析对应的谓词,确定外部输入(或一系列转换后的形式)触发安全相关 API 的取值范围,根据这一取值范围构造外部输入数据。在具体实现过程中,采用 Z3^[11]求解外部依赖。

2) 系统事件回调方法

如前文所述,系统事件回调方法可分为广播事件接收者、系统事件监听器、系统内容提供者回调方法这 3 类。对于系统广播事件回调方法,静态注册的广播接收者只需分析 AndroidManifest.xml 文件便可确定其依赖的系统事件;对于动态注册的广播接收者,需要分析其数据依赖关系,即分析 Intent 的属性,根据 data、action 等属性确定具体的系统事件。对于系统事件监听器,分析其对应的过程依赖子图的入口处监听器方法名,先确定系统事件类型;再根据源码中具体调用的成员方法,确定具体的系统事件。对于系统内容提供者监听器,分析其对应的控制依赖子图,确定监控数据改变监听点(即 registerContentObserver 方法),分析 registerContentObserver 的参数,确定具体监听的数据(数据采用 uri 描述)。由此,触

发系统内容提供者监听器所需的输入事件为对前步确定的数据进行插入、修改或删除操作。

2.3 生成输入

在得到触发回调方法依赖的 UI 输入和系统事件输入后,动态分析时采用算法 2 向模拟器中输入事件。样本的组件间调用图和样本中所有基本组件的系统依赖图集合作为算法输入,首先从组件间调用图的全部入口点中随机选择一个节点开始遍历。由于当前遍历的基本组件为组件间调用图的入口点(如 MainActivity 或系统广播接收者),Pre(前向触发依赖)赋值为空(算法 2 第 22、23 行),不需要外部事件触发;搜索系统依赖图集得到当前基本组件的依赖图(算法 2 第 33 行),并提取当前基本组件包含的 UI 事件回调方法、系统事件回调方法、同时包含前两类回调方法的特殊回调方法(算法 2 第 34 行)以及那些能够触发组件切换的跳转方法(Transitive_Call,算法 2 第 35 行)。得到目标回调方法后,按照如下方式生成输入事件:先输入那些不会触发组件切换的回调方法所需的事件,最后输入那些会触发组件切换的回调方法所需的事件。算法 2 采用的顺序为 UI 事件回调方法、系统事件回调方法和组合回调方法(算法 2 第 36、39、42 行)。输入事件生成在 InputEvent 过程完成,根据系统依赖图提取触发目标回调方法所需的组件内依赖后,组合组件外和组件内依赖得到最终需要的事件链。当所有目标回调方法都触发完毕后,触发跳转方法,跳转到下一个基本组件(算法 2 第 50~53 行),算法 2 执行流直接跳转到算法 2 第 11 行。

如前文所述,每个回调方法依赖的事件由组件外依赖 Pre 和组件内依赖 Con 这 2 部分组成。其中,Pre 与当前组件所有前驱(某次遍历)的跳转依赖事件相关,Con 与当前组件的功能逻辑相关。在 InputEvent 过程中,通过分析组件的系统依赖图提取 Con,将 Pre 与 Con 联合起来构造输入事件。Pre 与当前组件在当前遍历路径上的序号(入口点序号为 0,后续节点依此逐一递增)相关,由于采用深度优先遍历,设计了 Pre 栈 PreStack,每遍历一个基本组件后将其前身依赖和跳转到下一个基本组件所需的输入事件添加到 PreStack 中(算法 2 第 51、52 行),从而实现通

过 $Pre[level]$ 保存当前组件跳转到下一组件所需的输入依赖。入口点基本组件的序号为 0，每遍历一个新的基本组件时，首先判断其是否为入口节点。若是，则 Pre 为空；若不是，则先获取当前基本组件的序号，然后从 $PreStack$ 中获取跳转到当前基本组件所需的输入依赖（算法 2 第 25、26 行）。若当前基本组件为当前遍历路径上的最后一个节点，则移除 $PreStack$ 中最新加入的输入事件。

算法 2 事件输入算法

输入 CCG，组件间调用图

CSDG_Set，基本组件系统依赖图集合

输出 各类输入事件

```

1) Procedure InputEvent(sdg, callback, pre):
2)   con ← ExtractCon(sdg, callback)
3)   events ← {append pre and con}
4)   {向模拟器输入外部事件，程序跳转到下一个基本组件}
5) Output events, component
6) Entry_nodes ← {CCG 所有入口点}
7) Vertices ← {CCG 中所有节点}
8) While Vertices is not null:
9)   component ← RandomSelect (Entry_nodes)
10)  Entry_nodes ← {从 Entry_nodes 中删除 component}
11)  if component is not in Vertices:
12)    Next_component ← SearchUnexploredLeafNode(component)
13)    if next_component is null:
14)      if isEntry_nodes(component):
15)        go to line 8 //选择新入口点
16)      else
17)        {回退到前一个基本组件，更新当前基本组件}
18)        {go to line 11}
19)      end if
20)    end if
21)  end if
22)  if isEntry_nodes(component):

```

```

23)    pre ← null
24)  else:
25)    level ← {当前基本组件在当前遍历路径上的序号}
26)    pre ← PreStack[level]
27)    if isBottomLevelComponent(component):
28)      PreStack ← {移除 PreStack 中最新加入的输入事件}
29)      Flag ← True
30)    end if
31)    Flag ← False
32)  end if
33)  sdg ← CSDG_Set(component)
34)  UI, Sys, Combined ← Traverse(sdg)
35)  Transitive_call ← ContainTransferMethod(sdg)
36)  for item in UI – Transitive_Call do
37)    InputEvent(sdg, item, pre)
38)  done
39)  for item in Sys – Transitive_Call do
40)    InputEvent(sdg, item, pre)
41)  done
42)  for item in Combined – Transitive_Call do
43)    InputEvent(sdg, item, pre)
44)  done
45)  Vertices ← {从 Vertices 中删除 component}
46)  if Flag:
47)    {回退到前一个基本组件，更新当前基本组件}
48)    {go to line 11}
49)  end if
50)  for item in Transitive_Call do
51)    new_events, new_component ← InputEvent(sdg, item, pre)
52)    PreStack ← {append new_events to PreStack}
53)    {go to line 11}
54)  done

```

55) done

当遍历到当前路径上最后一个基本组件时,通过输入“回退”键返回上一级基本组件(算法2第27、29、46、47、48行)。这种做法只对目标组件为Activity时有效,当目标组件为Service或Broad Receiver时,无法通过“回退”键返回到当前组件。虽然可以通过重启应用或重装应用重新遍历到当前组件,但这种做法对时间性能影响太大。通过分析大量Android应用程序发现,绝大部分Service和Broadcast Receiver只包含一个Transitive_Call,因而上述问题对算法实际应用影响不大。

3 实验及结果分析

原型系统由Python实现,集成了apktool^[12]、Androguard^[13]和约束求解器Z3的部分功能,利用apktool解析样本,获取smali源码;利用Androguard的apk模块完成对样本的基本分析,获取样本的基本信息。对Androguard的analysis模块进行扩展、增加相关代码后,完成smali源码分析,构造组件间调用图和单个基本组件的系统依赖图;利用Z3求解安全相关回调方法的外部数据依赖。动态分析时,通过Android模拟器运行Android样本,通过Python脚本调用adb命令完成样本的安装、卸载、模拟器启动与还原、轨迹记录等功能。Android模拟器API版本为23,系统版本为6.0,型号为Nexus S,主要配置为1 024 M内存、64 M VM Heap、200 M的SD卡。

从已有数据集中随机选取1 000个普通应用和1 000个恶意应用作为实验数据集。分别采用方法覆盖率、权限方法覆盖率和基本组件覆盖率这3个指标检测本文方法的有效性。方法覆盖率指实验过程中样本执行的所有方法个数与样本方法总数的比值,权限方法覆盖率指实验过程中执行的权限方法数与样本权限方法总数的比值,基本组件覆盖率指实验过程中触发的组件数与样本总组件数的比值。

3.1 实验过程

本文采用与文献[14]相同的方法获取样本运行时的执行轨迹,利用“adb shell am start”启动应用,利用“adb shell ps | grep”获取应用对应的

PID,然后根据PID记录应用运行的trace。这种方法需要先启动应用获取其PID后,才能追踪应用运行时的trace。在应用启动过程中,MainActivity的onCreate、onStart和onResume方法会执行,这部分执行轨迹不会保存在最终的trace文件中,这样会降低分析结果的准确性。为此,对文献[14]中的方法做出如下改进:在最后计算覆盖率时,将上述3个方法调用的方法包含到动态执行的方法中。

在实验过程中,设计了3组实验,完成静态分析后,同时启动3个原始状态一致的模拟器,将样本分别安装到3个模拟器中,并同时进行动态分析。为方便与已有工作进行对比,将基准分析时间设置为180 s。与此同时,分析时间在基准时间上分别增加60 s和120 s,以对比分析时间增加对实验性能的影响。

1) 组1

组1利用Android官方提供的压力测试工具Monkey随机生成输入事件,每次模拟1 000个伪随机事件,并添加表4所列参数。SEED值取为500,throttle设为10 ms。

表4 Monkey 参数	
参数	说明
--ignore-security-exceptions	忽略安全异常
-s SEED	随机种子,与事件序列秩序相关
--throttle	事件间隔时间

2) 组2

组2利用MonkeyRunner生成输入事件,MonkeyRunner提供了API接口,支持用户编写Python脚本指导自动事件生成。分析AndroidManifest.xml文件,得到样本关注的所有系统事件,并增加对这些系统事件的模拟输入。

3) 组3

组3以本文方法的静态分析结果作为事件输入来源,利用开源包wrapEasyMonkey编写脚本程序生成事件输入,使用算法2向模拟器中输入事件。

3.2 结果分析

普通应用和恶意应用的平均方法数、平均权限方法数如图1所示。普通应用和恶意应用平均

权限方法个数基本接近,但普通应用平均方法数明显多于恶意应用,由此可知,恶意应用中权限方法数占总方法数的比例明显高于普通应用。普通应用与恶意应用各类组件平均数如图2所示,两类应用与无UI界面个数相差无几,而组件总数相差明显,其中,恶意应用无UI界面组件在所有组件中的比例为20.14%,而普通应用只有13.43%。恶意应用无UI组件率明显高于普通应用,这与恶意应用通常在后台完成恶意逻辑的情况一致。

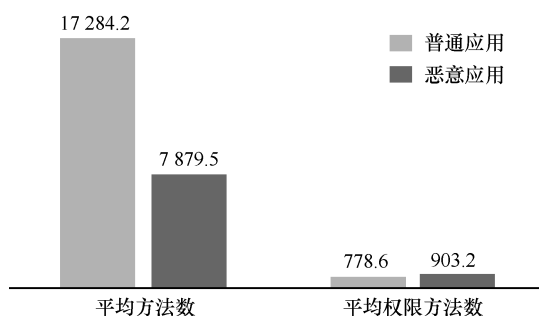


图1 普通应用与恶意应用方法数对比

3组实验的普通应用和恶意应用整体方法覆盖率、权限方法覆盖率和组件覆盖率如表5和表6所示。3种方法的整体方法覆盖率偏低,最高只有34.29%。这是由于样本中存在不可达代码、Java异常控制代码以及一些需要特殊操作才能遍历的代码。当分析对象为恶意应用时,实验性能较优,尤其是权限方法覆盖率和组件覆盖率优势较明显。这是由于大多数恶意应用喜欢借助无UI界面的后台服务或广播接收者完成恶意逻辑,而且这类后台行为通常是自启动或依赖敏感系统事件而被触发。正是由于这类后台自动执行代码现象的存在,提高了恶意应用在实验过程中的代码覆盖率、权限方法覆盖率和组件覆盖率。

利用Monkey生成输入时,由于不对样本进

行任何分析,缺乏关于样本功能逻辑的知识,盲目地输入无序伪随机事件,无法遍历大部分组件,因而触发包含权限方法的路径数非常有限。MonkeyRunner实验性能优于Monkey,这是由于MonkeyRunner提供了API接口,支持用户根据应用各个Activity的上下文,输入符合Activity功能逻辑的输入事件,使MonkeyRunner生成的事件输入能够遍历比Monkey生成的事件输入覆盖更多的执行路径。需要指出的是,MonkeyRunner只是根据Activity的View层级关系生成UI事件,缺乏对应用内部组件间调用关系和复杂回调方法所需UI事件、系统事件输入次序的分析,使MonkeyRunner难以触发那些依赖多个按序输入事件的功能逻辑,以及难以遍历某些应用组件。

本文方法实验性能最佳,当动态分析时间达到300s时,两类应用的权限方法覆盖率和组件覆盖率能够分别达到70%和65%以上。虽然本文方法的整体方法覆盖率在3组实验中最低,但权限方法覆盖率和组件覆盖率均优于基于Monkey和简单MonkeyRunner,说明本文方法能够在较低方法覆盖率的条件下,触发更多的权限方法和遍历更多的应用组件。这是由于本文方法针对安全相关回调方法提取其事件依赖,实际触发的程序执行路径具有单一性,执行的方法个数较少、但权限方法占有所有执行方法的比例较大;与此同时,本文方法根据组件间调用图和单个组件内部的系统依赖图,获取组件间跳转的事件依赖,因而能够比其他2种方法遍历更多的组件。对于整体方法覆盖率,由于本文方法设计的基本目标在于覆盖安全相关的回调方法,实验过程中覆盖的所有方法只是与安全相关回调在同一条执行路径上的方法,而其他不在同一条执行路径上的方法都不会覆盖,致使整体方法的覆盖率相比其他两组实

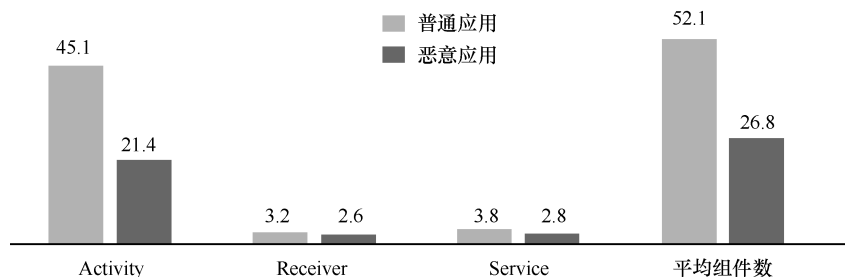


图2 普通应用与恶意应用组件数对比

表 5 普通应用实验结果对比

实验测试工具	整体方法覆盖率			权限方法覆盖率			组件覆盖率		
	180 s	240 s	300 s	180 s	240 s	300 s	180 s	240 s	300 s
Monkey	19.54%	27.12%	32.30%	10.09%	15.67%	19.37%	29.91%	35.63%	40.33%
MonkeyRunner	22.76%	29.58%	34.29%	16.72%	23.61%	28.92%	31.62%	42.56%	48.15%
本文方法	17.23%	24.75%	30.66%	50.13%	61.56%	71.22%	49.24%	60.87%	67.36%

表 6 恶意应用实验性能对比

实验测试工具	整体方法覆盖率			权限方法覆盖率			组件覆盖率		
	180 s	240 s	300 s	180 s	240 s	300 s	180 s	240 s	300 s
Monkey	20.68%	29.35%	33.12%	12.61%	17.53%	20.95%	31.27%	37.83%	42.19%
MonkeyRunner	24.56%	30.96%	36.18%	19.24%	26.37%	30.41%	34.29%	45.22%	50.71%
本文方法	18.97%	26.41%	32.52%	52.13%	63.56%	75.16%	53.24%	63.09%	70.67%

验低。需要指出的是,在利用算法 2 输入 UI 事件和系统事件时,状态回退是通过输入“回退”键实现的。当目标组件为 Service 或 Broadcast Receiver 时,无法回退到当前组件。这一局限使某些基本组件无法覆盖,从而影响了权限方法覆盖率和整体方法覆盖率。

3.3 时间性能

时间性能与分析环境配置、样本静态分析、事件输入速度和应用反应速度有关。事件输入速度能达到毫秒级,虽然事件之间存在缓冲间隔,但对时间性能影响相对较小。由于运行在基于 Qemu 的虚拟环境中,Android 应用的反应速度比真机要慢许多,能明显感觉到卡顿,延迟时间能达到秒级。分析环境配置和样本静态分析时间对时间性能影响最大。分析环境配置主要包括模拟器还原、安装样本(时间与样本大小正相关)这 2 个过程。第一次启动模拟器所需时间约为 30~40 s,快照还原模拟器和安装下一个样本,所需时间因样本大小而异,平均需要 35 s 左右;样本静态分析同样因样本而异,平均需要 180 s 左右。由此,整个事件输入前的准备工作所需时间在 215 s 左右,加上动态分析所需时间,分析一个样本所需的时间为 360~500 s。

4 相关工作

关于 Android 应用 GUI 测试,学术界提出许多研究方案^[15-17]。Yang 等^[18]提出一种灰盒 GUI

自动测试模型,首先利用 WALA 对 Android 应用源码进行静态分析,提取应用包含的 GUI 事件,并利用 Robotium^[19]生成并输入这些事件,再利用动态爬虫构造应用的 GUI 自动生成模型。Choi 等^[20]利用机器学习方法构造 Android 应用的 GUI 模型,基于应用 GUI 模型生成用户输入,遍历 Android 应用状态集中未遍历的状态,并提出 Swift-Hand 算法,能够根据应用执行轨迹动态修正 GUI 模型。Azim 等^[21]对 Android 应用字节码进行静态污点分析,构造包含 Activity 间跳转关系的控制流图,依赖此控制流图提出 Targeted Exploration 和 Depth-first Exploration 这 2 种 Android 应用 UI 事件自动生成策略。

Android 应用 GUI 测试的目标在于验证应用逻辑功能的正确性、检测潜在的软件缺陷,要求尽可能地遍历所有可能执行的路径,GUI 测试局限于 UI 事件,缺乏各类系统事件的分析。GUI 测试方案依赖于应用源码构造 GUI 模型,而检测 Android 恶意应用时,通常无法获取源码。

Appsplayground^[22]提出一种基于启发式的自动 GUI 遍历方法,以 Activity 作为应用的 GUI 状态,根据 Activity 的相似度避免重复遍历相同的 Activity,基于控件关联的文本、图片、层级关系等特征识别不同控件,并采用一种启发式的顺序输入策略生成每个 Activity 状态下的 GUI 事件。由于 Appsplayground 只对 Activity 遍历进行研究,分析的状态有限,无法分析系统事件相关的执行

路径。TraceDroid^[23]通过修改 Dalvik 虚拟机, 增加插桩模块, 能够记录 Android 应用运行时产生的动态执行轨迹。TraceDroid 采用简单的输入生成方法, 分别以 Monkey 和 MonkeyRunner 作为输入生成方法, 关注的是整体方法覆盖率, 缺乏如何触发安全相关路径的研究。

本文方法代码覆盖率与相关工作对比如表 7 所示, Appsplayground、TraceDroid、Andrubis^[24]、本文方法这 4 种方法的代码覆盖率均低于 40%, 这说明在时间一定、输入有限的条件下动态分析过程代码覆盖率普遍偏低。相比之下, 本文方法的恶意应用代码覆盖率高于其他 3 种方法, 普通应用的平均代码覆盖率一般。需要指出的是, 已有相关工作以整体代码覆盖率作为主要的性能指标, 而本文方法以权限方法覆盖率、基本组件覆盖率作为主要性能指标, 因而本文方法性能无法直接与其他相关工作进行对比。相比已有相关工作, 本文方法更细粒度地分析了代码覆盖问题, 从权限方法覆盖率和基本组件覆盖率这 2 个角度测试安全相关的代码覆盖率, 而且实验结果表明, 采用本文方法不管是权限方法覆盖率还是基本组件覆盖率都能达到 60%以上, 验证了本文方法的有效性。

表 7 相关工作性能对比

相关工作	普通应用代码覆盖率	恶意应用代码覆盖率	整体代码覆盖率
Andrubis	27.74%	27.80%	—
AppsplayGround	—	—	33%
Tracedroid	38.49%	27.61%	—
本文方法	30.66%	32.52%	—

5 结束语

针对现有 Android 恶意应用动态分析方法中, 如何生成有效输入事件触发安全相关行为方面存在的问题, 本文提出了一种基于静态分析的 Android 应用动态分析输入生成方法, 以权限方法覆盖率为目标, 构造组件间调用图获取应用的组件间跳转关系, 构造单个组件的系统依赖图, 并采用程序切片方法, 获取触发组件内部回调方法和组件跳转所需的事件依赖, 并设计事件输入

算法。实验结果表明, 本文方法能够有效提高权限方法的覆盖率。下一步工作包括: 解决回调方法遍历过程中状态回退时部分组件无法遍历的问题; 进一步优化事件输入算法, 提高权限方法覆盖率以及时间性能。

参考文献:

- [1] XIA M, GONG L, LYU Y, et al. Effective real-time android application auditing[C]//IEEE Symposium on Security and Privacy. 2015:899-914.
- [2] BIANCHI A, FRATANTONIO Y, KRUEGEL C, et al. NJAS: sandboxing unmodified applications in non-rooted devices running stock Android[C]//ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. 2015:27-38.
- [3] BACKES M, BUGIEL S, HAMMER C, et al. Boxify: full-fledged app sandboxing for stock android[C]//Usenix Conference on Security Symposium. 2015:691-706.
- [4] AFONSO V M, AMORIM M F D, GRÉGIO A R A, et al. Identifying Android malware using dynamically obtained features[J]. Journal of Computer Virology and Hacking Techniques, 2015, 11(1): 9-17.
- [5] QIU L, ZHANG Z, SHEN Z, et al. AppTrace: dynamic trace on Android devices[C]//IEEE International Conference on Communications. 2015:7145-7150.
- [6] Monkey[EB/OL].<https://developer.android.com/studio/test/monkey.html>.
- [7] MonkeyRunner[EB/OL].<https://developer.android.com/studio/test/monkeyrunner/index.html>.
- [8] 李必信, 李宣东, 郑国梁. 一种系统依赖图的面向对象扩充方案[J]. 软件学报, 2001, 12(2): 241-248.
LI B X, LI X D, ZHENG G L. An object-oriented extension approach for system dependent graph[J]. Journal of Software, 2001, 12(2): 241-248.
- [9] 杜林, 江海燕. 计算面向对象程序切片技术研究[J]. 山东大学学报(工学版), 2008, 38(6):41-47.
DU L, JIANG H Y. Research on slicing technology of object oriented program[J]. Journal of Shandong University(Engineering Science), 2008, 38(6):41-47.
- [10] AU K W Y, ZHOU Y F, HUANG Z, et al. PScout: analyzing the Android permission specification[C]//The ACM Conference on Computer and Communications Security. 2012: 217-228.
- [11] MOURA L D, BJORNER N. Z3: an efficient SMT solver[J]. Lecture Notes in Computer Science, 2008, 4963:337-340.
- [12] Apktool[EB/OL]. <http://www.softpedia.com/get/Programming/Debuggers-Decompilers-Dissassemblers/ApkTool.shtml>.
- [13] Androguard[EB/OL].<https://github.com/androguard/androguard>.
- [14] CHOI S, BIJOU M, SUN K, et al. API tracing tool for Android-based mobile devices[J]. International Journal of Information and Education Technology, 2015, 5(6):460-465.
- [15] MACHIRY A, TAHILIANI R, NAIK M. Dynodroid: an input generation system for Android apps[C]//Joint Meeting on Foundations of Software Engineering. 2013:422-434.
- [16] HU C, NEAMTIU I. Automating GUI testing for Android applications[C]//The International Workshop on Automation of Software

Test. 2011:77-83.

- [17] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. Using GUI ripping for automated testing of Android applications[C]//The 27th IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society. 2012: 258-261.
- [18] YANG W, PRASAD M R, XIE T. A grey-box approach for automated GUI-model generation of mobile applications[C]//The International Conference on Fundamental Approaches To Software Engineering. 2013:250-265.
- [19] Robotium[EB/OL].[tps://github.com/RobotiumTech/robotium](https://github.com/RobotiumTech/robotium).
- [20] CHOI W, NECULA G, SEN K. Guided GUI testing of Android Apps with minimal restart and approximate learning[J]. ACM Sigplan Notices, 2013, 48(10):623-640.
- [21] AZIM T, NEAMTIU I. Targeted and depth-first exploration for systematic testing of Android Apps[J]. ACM Sigplan Notices, 2013, 48(10):641-660.
- [22] RASTOGI V, CHEN Y, ENCK W. Appsplayground: automatic security analysis of smartphone applications[C]//ACM Conference on Data and Application Security and Privacy. 2013:209-220.
- [23] VEEN V V D. Dynamic analysis of Android malware[J]. Internet & Web Technology Master thesis, VU University Amsterdam, 2013.
- [24] LINDORFER M, NEUGSCHWANDTNER M, WEICHSELBAUM L, et al. ANDRUBIS--1,000,000 apps later: a view on current Android malware behaviors[C]//The 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, IEEE Computer Society. 2014:3-17.



周振吉（1984-），男，江苏连云港人，博士，解放军理工大学讲师，主要研究方向包含协议逆向、移动应用安全。



洪征（1978-），男，江西上饶人，博士，解放军理工大学副教授，主要研究方向为协议逆向、网络安全。



颜慧颖（1993-），女，江西吉安人，解放军理工大学硕士生，主要研究方向为移动应用安全。

作者简介:



叶益林（1987-），男，江西九江人，解放军理工大学博士生，主要研究方向为移动应用安全。



吴礼发（1968-），男，湖北仙桃人，解放军理工大学教授、博士生导师，主要研究方向包含移动应用安全、协议逆向、网络安全。