文章编号:2095-6134(2018)03-0409-08

一种自动化的 Android 应用定向行为测试方法*

叶延玲[1,2†],傅晓彤[1],张玉清[2],乐洪舟[2]

(1 西安电子科技大学网络与信息安全学院,西安 710071;
2 中国科学院大学国家计算机网络入侵防范中心,北京 101408)

(2017 年 1 月 13 日收稿;2017 年 4 月 21 日收修改稿)

Ye Y L, Fu X T, Zhang Y Q, et al. An automated and directed testing technique for target behavior of Android application[J]. Journal of University of Chinese Academy of Sciences, 2018,35(3):409-416.

摘 要 Android 应用特定行为的定向测试通常被用来测试应用是否存在隐私泄漏、远程控制等恶意行为。为解决已有定向测试方法存在的失败率高和耗时多等问题,提出一种以目标 API 调用代表程序特定行为的自动化测试方法。首先,用静态分析得出到达目标 API 调用位置的路径;然后在动态测试过程中,排除无关组件和控件,使应用沿路径自动运行至目标 API 调用的位置,触发特定行为。实验证明,本方法完成 Android 应用定向行为测试的效率较高。

关键词 Android;定向测试;目标 API;静态分析;动态测试

中图分类号:TP393 **文献标志码**:A **doi**:10.7523/j.issn.2095-6134.2018.03.016

An automated and directed testing technique for target behavior of Android application

YE Yanling[1,2], FU Xiaotong[1], ZHANG Yuqing[2], YUE Hongzhou[2]

(1 *School of Cyber Engineering, Xidian University, Xi'an* 710071, *China*;
2 *National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing* 101408, *China*)

Abstract Directed testing of specific behaviors for Android applications is usually used to detect privacy leak, remote control, or other malicious behaviors. In order to solve the problems of the high failure-rate and large time-consuming of the present approaches, an automated testing method that uses target API invocation to represent the application's behavior is proposed. First, the method gets the invocation paths to the target API by using static analysis. Then dynamic testing is adopted to exclude extraneous components and GUI elements, and the application is driven to automatically run along the specific paths to reach the target API invocations and the specific behavior is triggered. Experimental results show that the method achieves a high efficiency.

Keywords Android; directed test; target API; static analysis; dynamic testing

* 国家重点研发计划(2016YFB0800700)、国家自然科学基金(61572460, 61272481)、国家发展和改革委员会国家信息安全专项((2012)1424)和信息安全国家重点实验室开放基金(2017-ZD-01)资助

† 通信作者,E-mail:yeyl@nipc.org.cn

由于 Android 系统自身的开放性,应用市场审核标准不一,导致一些开发者弱化对应用(App)安全性的考虑,为了获取用户数据申请不必要的敏感权限,调用敏感 API,如读取手机 IMEI 码、读写联系人和通话记录等。这些敏感行为可能导致隐私泄露[1]、诱骗欺诈等问题,给用户造成巨大损失。作为用户下载应用的来源,应用市场应该保证应用的安全性,测试应用是否存在敏感行为。此外,随着应用数量的快速增长,测试者对自动化测试方法的效率提出了更高的要求。目前已提出的自动化动态测试方法主要有两种:全覆盖测试和定向测试。在测试 App 的特定行为时,基于全覆盖方法的测试工具会造成大量的时间和资源浪费,而定向测试的方法由于更具目的性和方向性,能够极大地提高效率。

定向测试方法大多采用以静态分析辅助动态测试过程的方法,依靠对 App 代码静态分析得到的信息,以尽可能少的输入事件到达目标位置。目前,测试 App 敏感行为的相关工具主要有 IntelliDrod[2]、Brahmastra[3] 和 SmartDroid[4] 等,但它们的目标各不相同。IntelliDrod 依靠精确的静态分析在动态测试生成特定输入,提高动态测试工具测试操作的有效性。Brahmastra 主要用于检测恶意应用的第三方方法调用,通过重写 App 代码控制 App 状态跳转,提高测试效率。SmartDroid 能够自动化获取触发敏感 API 的条件集,动态测试工具与之结合后能够自动化测试 App 敏感行为。在进行定向行为测试时,这些方法存在一些限制:1)工具本身不能测试 App 敏感行为,需要结合其他动态测试工具;2)对静态分析结果的精确度要求过高或对 App 代码重写,导致使用工具测试时失败的可能性较大。

为解决上述问题,本文提出一种基于 GUI 的自动化测试方法,对 App 的特定行为进行定向触发。首先,将该行为对应的 API 设为目标 API,对 App 进行静态控制流分析和污点分析,获取到达目标 API 调用位置的路径上的有效组件和控件事件处理方法,形成一条调用路径。当静态分析失败或有误差时,进入误差控制模块。然后,在动态测试过程中,通过实时检测当前页面状态,并实时读取系统日志输出,确保只访问有效组件和控件,使 App 沿着路径运行,直到抵达路径终点。

本文的方法在不需要对 App 进行任何修改的情况下实现自动化定向测试,同时不过于依赖静态分析结果,避免了测试失败率高的问题。在测试 App 特定行为时,能够排除大量 Activity 和控件,使自动化测试更具目标性和方向性。经实验验证,该方法到达目标 API 调用位置所需操作的控件数平均是全遍历和随机触发方法所操作控件数的 30.7% 和 13.0%,有效地提高了测试效率。并对 61 个目标 API 进行测试,其中只有 3 个测试失败,失败率较低。

1 相关工作

Android 平台上的 App 自动化测试工具主要分为全覆盖测试和定向测试两种。全覆盖自动化动态测试工具需要尽可能覆盖所有控件,在测试 App 特定行为时效率较低。由 Google 发布的 Monkey[5] 可以随机向目标程序发送事件流和数据流,在其基础上开发的框架 PUMA[6] 可用于压力测试。MonkeyRunner[7]、Robotium[8] 与 UiAutomator[9] 都是 Android SDK 工具包提供的辅助测试工具,但不能完全自动化。A3E[10]、AppsPlayground[11]、Sapienz[12]、AndroidRipper[13] 等工具以及 Dutia 等[14] 设计的方法致力于通过构建 GUI 模型,设计自动化生成正确输入格式等方式来提高全遍历控件时的控件覆盖率。Amalfitano 等[15] 提出的快速碰撞测试和回归测试方法,可自动建立 GUI 模型并生成自动执行的测试用例。赵耀宗等[16] 通过模拟用户行为的方法自动探测 GUI,能够以较高的覆盖率遍历 GUI 控件。BugRocket[17] 建立分布式测试系统,能够自动化生成测试脚本,并将测试请求分发给测试云上的设备。PATS[18] 构建细粒度模型,并行化执行测试过程,极大地提高了测试效率。MobiGUITAR[19] 通过遍历 App GUI 创建状态机模型,并据此生成和执行测试用例。Dynodroid[20] 把 App 视为与运行环境交互的事件驱动程序,监控 App 对每个事件的反应,并据此生成下一个事件。它提供 UI 和系统事件输入,并能够把机器事件和人工事件无缝结合,提高了代码覆盖率。

定向测试 App 特定行为的工具大多使用静态分析与动态分析相结合的方法。SmartDroid 首先利用静态分析得到运行至敏感 API 的 Activity,遍历这些 Activity 上的所有控件时监控日志输出的 GUI 信息并进行分析,得到到达该敏感 API 的有效 UI。Brahmaster 通过静态分析获得到达第三方组件中的 API 调用路径,然后对 App 进行代码

重写,控制 App 运行时的跳转,使其到达敏感 API 调用位置。然而对 App 代码重写存在重签名失败、App 运行时崩溃等问题,可能导致测试失败。IntelliDroid 在分析出目标 API 的调用路径后,提取控制目标 API 调用的条件,构造特定输入值,提高动态测试工具触发目标 API 的效率。由于 IntelliDroid 生成的输入会注入所有能够触发目标 API 的事件处理器,因此静态分析阶段的结果必须非常精确,否则生成的输入值会发生错误,导致最终无法到达目标 API 调用位置。

2　设计与实现

本文方法针对 App 特定行为进行定向测试,在分析过程中以目标 API 近似代表程序行为。如 App 打开设备摄像头这一特定行为,在代码中调用 < android. hardware. Camera: open( ) > 方法,在分析时把此方法定为目标 API。

方法分为静态和动态 2 个分析模块。静态分析模块获取到达目标 API 调用位置可能经过的组件和控件事件处理方法,以引导动态过程,提高测试效率。当静态分析有误时,进入误差控制过程,减少组件排除量,防止动态测试失败。为了监控控件事件,对 Android 源码进行修改,使得每次调用回调方法都有日志输出。在动态模块中,操作控件并监控 Activity 跳转和系统日志输出,以尽可能少的控件操作驱动 App 运行至目标 API 位置。

2.1　静态分析模块

静态分析能够得到引导动态测试过程的信息,即从主 Activity 到目标 API 调用位置的组件跳转路径。编译过的源码文件——DEX 文件是静态分析的对象。本文选择使用开源框架 Soot[21] 对其进行静态分析,以及在 Soot 基础上开发出的 FlowDroid[22] 对 Android 程序进行污点分析[23]。FlowDroid 在对 App 进行污点分析时先使用 IFDS[24] 模型得到一个高效的虚拟 main 方法,使 App 中的所有组件的生命周期方法和回调方法可以按照任意顺序调用,用于生成调用图和进程间控制流图(ICFG)。污点分析的过程从检测到的污染源开始,通过遍历 ICFG 追踪被污染的数据,到达陷入点后污点分析结束,并输出从污染源到陷入点的路径。

App 运行至目标 API 调用位置的过程中可能需要启动 Activity、Service、BroadcastReceiver 等组件。在用 FlowDroid 分析前将定义 Intent 的语句设为 Source,并把表 1 所示的启动新组件的方法设置为 Sink。使用 FlowDroid 进行污点分析,得到组件间的启动关系,例如,Activity A 中的一个回调方法 onClick( ) 调用 startActivity( ) 方法启动 Activity B。这一过程只能得到通过调用表 1 中的方法显式启动的组件,因此,还需要使用 Soot 解析 APK 文件,得到组件的隐式启动关系,如匹配 Intent 属性启动 Activity。

表 1　组件启动方法

Table 1　Method for starting a component

组件	启动方法
Activity	startActivity startActivities startActivityForResult startActivityFromChild startActivityFromFragment startActivityIfNeeded
Service	startService bindService
BroadcastRe-ceiver	sendBroadcast sendBroadcastAsUser sendOrderedBroadcast sendOrderedBroadcastAsUser sendStickyBroadcast sendStickyBroadcastAsUser sendStickyOrderedBroadcast sendStickyOrderedBroadcastAsUser

由于一个 Activity 可能对应多个 GUI 界面,需要在动态分析阶段监控回调方法的调用以排除无关 GUI 控件,因此,还需解析得到调用组件启动方法的回调方法。将组件信息、回调方法以及启动的组件信息结合,形成路径节点,解析的关键程序代码和得到的节点信息举例如图 1 所示。

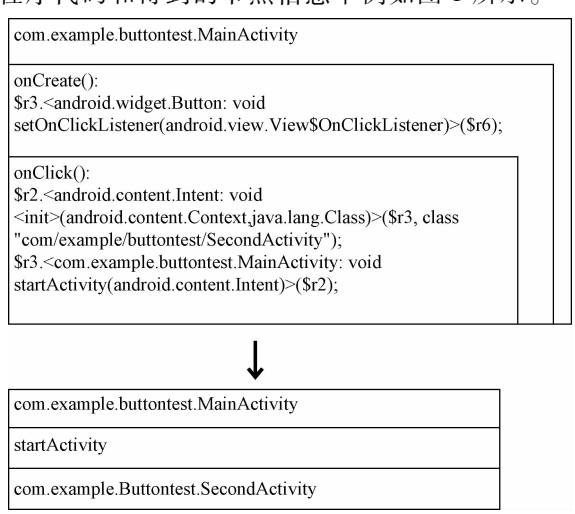

图 1　解析代码得到节点信息

Fig. 1　Parsing codes for getting node information

通过解析 DEX 文件,还能够得到调用目标 API 的组件,即为最终组件。同时,最终组件调用目标 API 的方法也可获得。目标 API 可能被最终组件的 3 种方法调用:第 1 种,某控件绑定的监听方法调用的回调方法,操作该控件即可触发目标 API,将该回调方法及其类名信息加入路径的终节点,由该类名可得到触发该回调方法的控件所在的 Activity(但并不知道控件是哪一个);第 2 种,组件的生命周期方法,启动该组件后即可触发目标 API,则该组件即为终点;第 3 种,普通的方法,但此方法最终由前面两种中的某一种调用。

分析结束后,把静态分析得到的各组件间显式启动和隐式启动关系从最终组件开始进行逆向分析,得到一条从主 Activity 到目标 API 调用位置的路径。具体过程举例如图 2 所示,图中字母表示组件,E 为最终组件,A 为主 Activity,箭头表示启动关系。若路径终点只有 Activity 信息,表示终点 Activity 打开时就已触发目标 API;若路径终节点含有回调方法信息,则说明目标 API 需要控件触发,那么在动态测试时,App 跳转至终点 Activity 后仍需遍历 GUI 控件,直到系统输出的日志与路径终点的回调方法信息一致。

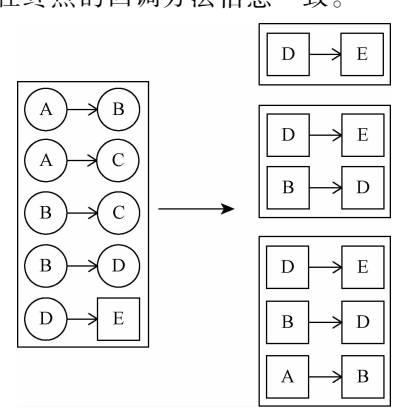

图 2 路径生成过程

Fig. 2 Process of path generation

2.2 误差控制

由于存在少数静态分析阶段无法分析出路径或路径有误差的情况,导致按照既定方法无法触发目标 API。为提高测试成功率,本文方法增加误差控制功能。在这种情况下,方法利用解析代码得到的 Activity 的信息把所有 Activity 分为3 类:1)确定与目标 API 相关;2)确定与目标 API 无关,即此 Activity 不能跳转至其他 Activity;3)其他。测试时,把除第 2)类之外的 Activity 全部加入路径,在动态测试过程中作为监控对象,判断 App 运行状态,其他过程按照正常动态遍历过程进行,以降低控件排除比例的方式提高测试的成功率。

2.3 动态分析模块

2.3.1 App 状态监控模块

由于静态分析不能得到当前 GUI 控件与回调方法的对应关系,因此,本文的动态分析过程需要自上而下触发当前 GUI 的可操作控件。方法采用修改 Android 源码的方法监控 App 状态。在遍历当前 GUI 的控件时,实时监控系统日志,并将其与路径中的回调方法信息匹配,以判断此次控件操作事件是否触发了路径中的回调方法。由于所有的 Android App 使用的回调机制都相同,本文的方法只需要修改 1 次 Android 源码即可。

App 运行中启动的多个 Activity 按照启动顺序依次以栈结构存储在内存中。使用 adb shell 命令即可获取栈顶 Activity,即当前与用户交互的 Activity。比较控件操作前后的栈顶 Activity 即可判断 Activity 是否发生了跳转,并依此监控 App 是否沿路径运行。操作某控件后,如果检测到 Activity 跳转,且该 Activity 不在路径中,则此 Activity 中的控件均无需访问。

控件操作能够使 GUI 在 Activity 不变的情况下发生改变,可能是控件属性改变或转换至另一个 GUI。只判断 Activity 的跳转可能导致控件操作无效或漏掉关键控件,影响测试效率。因此,必须在每次控件操作之后判断 GUI 是否发生改变。本文利用 UiAutomatorViewer 工具获取当前 GUI 的页面布局信息。首先,使用“adb shell sdkpath \\tools\\uiautomator name-compressed/path/name.xml”[25]命令把当前用户页面中控件的结构关系和属性等布局信息存储在 XML 文件中。然后解析该 XML 文件获取控件的可操作性、位置、文本信息以及页面结构等信息,形成以控件为节点的树结构(本文中称为控件树),并为每一个节点设置 weight(权值)属性。从叶子节点到根节点的每个节点权值设置为其所有子节点权值与该节点初始权值之和。以图 3 所示的页面布局的树结构为例说明自下而上计算权值的过程。用根节点权值代表该 GUI 权值。若控件操作前后根节点权值不变,说明 GUI 没有变化;反之,GUI 发生了转换。

2.3.2 控件操作过程

由于 Android App 运行过程中的 Activity 跳转是不可预测的,在控件遍历过程中必须把已遍历过的 GUI 做标记,并记录在已标记队列中。若

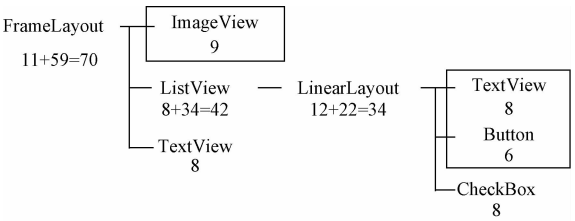

图 3　GUI 控件树举例

Fig. 3　Example of GUI widget tree

检测到此 GUI 已标记,则回退至上一界面,并触发下一未标记的可操作控件。若同级节点已全部标记,则将它们的父节点直接标记为已操作。自下而上进行标记,直到根节点标记为已操作,那么将被标记的根节点加入已标记队列,说明此 GUI 中控件已全部遍历。以图 3 中的 CheckBox 为例说明对控件树的标记和操作过程,方框中的控件为已操作。首先,将 CheckBox 标记为已操作,可知它的同级节点全部为已操作,那么将其父节点 LinearLayout 标记为已操作;同理,将 ListView 也标记为已操作;此时第二级节点只有 2 个被标记,则不标记根节点,且此 GUI 不加入已标记队列。

遍历控件树时采用深度优先遍历的方式,自上而下、自左至右地进行。对于控件树的某一节点,首先判断其是否有子节点,如果有,则对其子节点进行分析;若没有,将该控件标记为已操作,并按照算法 1 所示的过程对控件进行操作。当 App 运行至路径终点 Activity 时,若终节点不含回调方法,说明此时目标 API 已触发,测试结束;若路径终节点含有回调方法,则遍历此页面控件,直到系统输出含有此回调方法信息的日志,说明目标 API 已经被调用,测试结束。

Algorithm 1　widget operation Algorithm

```
Input: the widget in widgets tree of current GUI
1: if the widget is clickable then
2:         click the widget
3:         if Activity is changed then
4:             if the new Activity is the next
               node of the path then
5:                 get new GUI
6:                 traverse the widgets tree of
               the new GUI
7:             else
8:                 rollback
9:                 jump to line 16
10:        end if
11:        else
12:            if the log from system matches with the path then
13:                if GUI is changed then
14:                    jump to line 5
15:                else
16:                    operate the next widget
17:                end if
18:            else
19:                if GUI is changed then
20:                    jump to line 8
21:                else
22:                    jump to line 16
23:                end if
24:            end if
25:        end if
26:else
27:    jump to line 16
28:end if
```

3　实验

选取 Android 官方定义的危险权限[26]和一些特殊权限对应的行为进行测试,验证本文方法对于排除无关 Activity 和控件的有效性。为评估本文方法的性能,将实验结果与全遍历和随机触发的效率进行比较,并与其他相关方法进行优劣对比。

3.1　测试环境及数据

本文实现的工具运行在 Windows 7 系统下、内存为 8 G 的 PC 平台上,测试的 App 运行在系统为 Android 5.1.1、内存为 2 G 的测试机上。从 Google Paly 的流行应用中随机选取 5 个 App 的行为进行测试。这 5 个 App 普通且具有代表性,能够形象化地展示本文方法的效率和有效性。测试的 App 的特定行为主要有启动相机、读取 IMEI 码、NFC 通信、读/写联系人、杀死后台进程、连接网络、发送短信、获取 Wi-Fi 状态等,并把与这些行为相关的 API 调用设定为目标 API。

文献[10]中的数据表明,动态分析部分需要花费的时间平均是静态分析的 84.3 倍,因此,本实验忽略静态分析耗时对最终结果的影响,只分析动态部分的效率。实验结果如表 2 所示,排除的 Activity 指在到达目标 API 调用位置前的动态测试过程中排除的无效 Activity;排除的控件数包括无效 Activity 中的控件数和有效 Activity 中的无效控件数。动态点击的步数仅计算触发事件处理方法的点击事件次数,不考虑回退和滑动屏幕等操作,因此,到达目标 API 调用位置的步数即为本方法触发的控件数。

对于不同 App 和同一 App 的不同目标行为,由于目标 API 的调用位置各不相同,需要操作的控件数可能差别很大,但总体而言,遍历过程中操作的控件数远少于 App 的控件总数。因此,本方法的组件和控件排除极大地提高了测试效率。实验测试 61 个目标 API,16 个不能动态触发。其中 13 个目标 API 在 App 使用的第三方库中,但由于 App 本身并未调用,能够分析出 API 所在位置,不能动态触发。另外,有 3 个目标 API 由于静态分析出错而不能触发。

表 2 目标 API 测试结果

Table 2 Results of target API testing

包名	Activity 数	控件数	特定行为	点击的步数	排除的 Activity 数	排除的控件数
ApiDemos	351	1 344	启动相机	112	145	387
			连接网络	88	128	296
			NFC 通信	121	186	456
			读/写联系人	38	57	157
			读取 IMEI	127	189	461
			发送短信	130	192	467
com. ceic. App	17	30	连接网络	6	2	9
				16	4	13
com. mylgy. saomabijia	21	32	启动相机	7	3	15
			访问网络上的信息	5	1	11
			获取 Wi-Fi 状态	9	5	19
com. quyugongzuoshi. jinangwengongju	28	26	杀死后台进程	5	1	10
				6	3	12
tal. android. ChaChaKan	43	81	启动相机	6	1	4
			获取 Wi-Fi 状态	7	4	18
			连接网络	15	8	51

3.2 方法性能分析

本文的方法在触发各个目标 API 调用前访问的控件数占总控件数的比例如图 4 所示(图 4 ~ 图 6 中横轴为表 2 中的各目标 API)。对于实验中所有的目标 API,测试时点击的控件数平均是 App 中的所有控件数的 16.4% 。在测试目标 API 时,全遍历方法需要遍历所有组件,触发了路径外的组件和控件。对于随机控件触发方法,本文使用 Monkey 工具进行实验。测试结果的对比如图 5 所示,到达目标 API 位置前,本文方法需要的步数比随机触发的方法平均减少 87.0% 。本文与全遍历方式对比的情况如图 6 所示,本方法遍历的 Activity 与其相比平均减少 69.3% ,有效地避免了测试时间的浪费。由于动态测试的效率对整个测试过程影响最大,因此,大量无关 Activity 和控件的排除,使得本文提出的定向测试方法具有很高的效率。

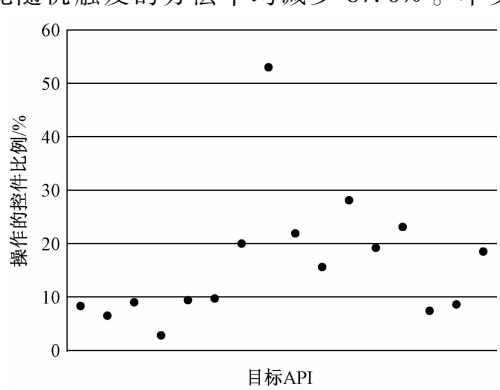

图 4 操作的控件比例

Fig. 4 Proportions of operated widgets

3.3 与定向测试相关工作的对比

分别提取 IntelliDroid、Brahmaster 与 SmartDroid 3 个方法进行对比,结果如表 3 所示。由表中可以看出,本文的方法有效规避了定向测试工具对静态分析的准确性依赖程度高、动态测试效率低等问题。Brahmaster 直接修改 App 组件启动属性,改变 App 状态,控制 App 跳转至第三方代码,测试效率较高,由此导致的重签名失败、运行时崩溃等问题使得测试失败率较高。IntelliDroid 用于自动生成能够驱动 App 运行的输入事件。为了快速触发目标 API,每一个输入都必须准确,使得测试的成败依赖于静态分析结果的正确性,该方法的失败率为 6.7% 。由于 App 运行中 Activity 跳转的不确定性,本文方法的输入事件同时基于静态分析结果和 App 运行状态进行判断,使得方法失败率较低,仅为 4.9% 。但

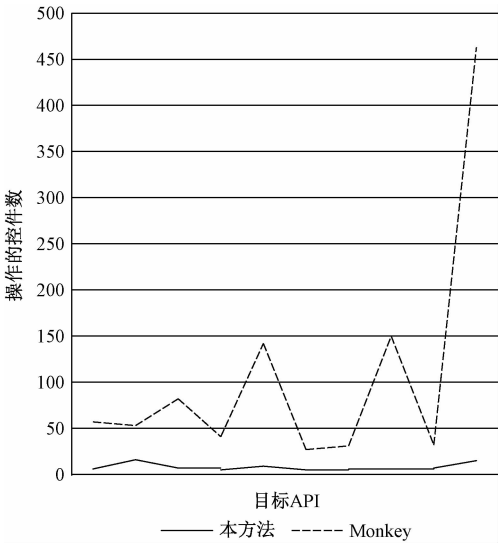

图 5　本方法与 Monkey 的操作控件数对比

Fig. 5　Comparison of operated button numbers between the two algotrithms

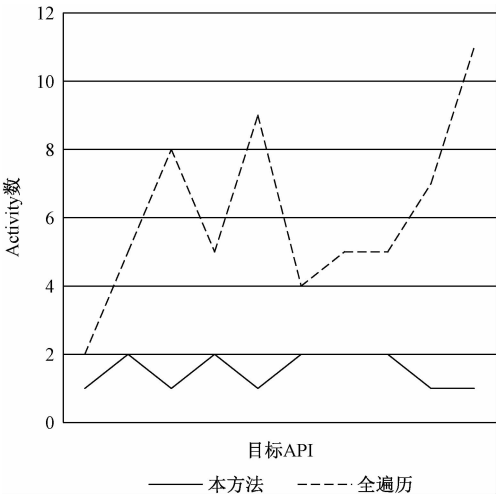

图 6　本方法与全遍历方法的遍历 Activity 数对比

Fig. 6　Comparison of traversed Activity numbers between the two algotrithms

表 3　相关定向测试工具特点对比

Table 3　Characteristics of related directional testing tools

方法名	静态部分	动态部分	优点	缺点	目的
SmartDroid	得到与目标 API 相关的组件和方法	自动化触发每个 UI 控件	生成 UI 触发条件,使动态测试方法自动化	效率较低	生成触发敏感 API 的条件集
Brahmaster	得到到达目标 API 的执行路径	重写代码,控制 App 跳转	效率高	失败率高	检测第三方代码中的敏感 API
IntelliDroid	得到精确的调用路径,提取控制条件	人工或自动化生成输入事件,驱动 App 运行	得到精确的输入事件,提高动态操作有效性	失败率高	生成针对动态测试的特定输入
本文方法	得到相关组件和控制事件处理方法	自动化触发部分控件	效率高,失败率低	输入事件类型较少	测试 APP 特定行为

是,与 IntelliDroid 相比,本文方法对于 App 运行时的输入事件类型考虑较少。SmartDroid 通过遍历有效 Activity 中的所有控件得到到达敏感 API 的 UI 控件集合,用以辅助动态分析工具进行 App 敏感行为测试。而本文方法通过排除无关组件以及标记关键回调方法,在动态分析阶段只需遍历部分 GUI 中关键控件之前的部分控件,即可测试 App 的特定行为,测试效率较高。

4　总结

本文提出一种针对 Android 应用特定行为进行自动化定向测试的方法。该方法将静态分析与动态分析相结合,使用静态解析 APK 文件得到的信息引导动态测试过程,以尽可能少的操作次数触发目标 API。通过实验及与其他方法的对比,验证了此方法能够有效提高 App 定向行为测试的效率。

未来的工作主要针对静态和动态两个模块进行。静态分析方面:当 App 比较复杂时分析结果可能有误差,从而影响测试结果。考虑改进静态分析算法,提高结果的准确率。动态分析方面:动态分析过程中频繁地获取当前 GUI 的控件信息,并将其以 XML 文件的方式从手机端发送至计算机端进行解析。这一过程对动态分析的速度影响较大,考虑改进控件信息获取和解析算法,提高计算效率。

参考文献

[1] 吴敬征,武延军,武志飞,等. 基于有向信息流的 Android 隐私泄露类恶意应用检测方法 [J]. 中国科学院大学学报, 2015, 32(6): 807-815.

[2] Wong M Y, Lie D. IntelliDroid: a targeted input generator for the dynamic analysis of Android malware [C] // Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS), 2016: 21-24.

[3] Bhoraskar R, Han S, Jeon J, et al. Brahmastra: driving apps to test the security of third-party components [C] // The

Usenix Security Symposium, 2014:1 021-1 036.

[4] Zheng C, Zhu S, Dai S, et al. Smartdroid: an automatic system for revealing ui-based trigger conditions in android Applications [C] // Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. ACM, 2012: 93-104.

[5] Android Developers. UI/Application exerciser monkey [EB/OL]. (2016-09-20) [2017-01-12]. https://developer.android.com/studio/test/monkey.html.

[6] Hao S, Liu B, Nath S, et al. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile Apps [C] // Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services. ACM, 2014: 204-217.

[7] Android Developers. Monkeyrunner [EB/OL]. (2016-09-20) [2017-01-12]. https://developer.android.com/studio/test/monkeyrunner/index.html.

[8] Google Code. Robotium [EB/OL]. (2017-01-11) [2017-01-12]. https://robotium.com/.

[9] Android Developers. Uiautomator [EB/OL]. (2016-05-14) [2017-01-12]. http://developer.android.com/tools/help/uiautomator/.

[10] Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of android Apps [J] // ACM SIGPLAN Notices, 2013, 48(10): 641-660.

[11] Rastogi V, Chen Y, Enck W. AppsPlayground: automatic security analysis of smartphone applications [C] // Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy. ACM, 2013: 209-220.

[12] Mao K, Harman M, Jia Y. Sapienz: multi-objective automated testing for android applications [C] // Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, 2016: 94-105.

[13] Memon A, Banerjee I, Nagarajan A. GUI ripping: reverse engineering of graphical user interfaces for testing [C] // Reverse Engineering, 2003. Wcre 2003. Proceedings. Working Conference on. IEEE, 2003:260-269.

[14] Dutia S N, Oh T H, Oh Y H. Developing automated input generator for android mobile device to evaluate malware behavior [C] // Proceedings of the 4th Annual ACM Conference on Research in Information Technology. ACM, 2015: 43.

[15] Amalfitano D, Fasolino A R, Tramontana P. A gui crawling-based technique for android mobile application testing [C] // 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2011: 252-261.

[16] 赵耀宗, 程绍银, 蒋凡. Android 应用程序 GUI 遍历的自动化方法 [J]. 计算机系统应用, 2015, 24(9): 219-224.

[17] Ma X, Wang N, Xie P, et al. An automated testing platform for mobile applications [C] // 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2016: 159-162.

[18] Wen H L, Lin C H, Hsieh T H, et al. PATS: a parallel GUI testing framework for android applications [C] // Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. IEEE, 2015, 2: 210-215.

[19] Amalfitano D, Fasolino A R, Tramontana P, et al. MobiGUITAR: automated model-based testing of mobile apps [J]. IEEE Software, 2015, 32(5): 53-59.

[20] Machiry A, Tahiliani R, Naik M. Dynodroid: an input generation system for android apps [C] // Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 224-234.

[21] Bartel A, Klein J, Le Traon Y, et al. Dexpler: converting android dalvik bytecode to jimple for static analysis with soot [C] // Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis. ACM, 2012: 27-38.

[22] Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps [J]. ACM SIGPLAN Notices, 2014, 49(6): 259-269.

[23] 孔德光, 郑烇, 帅建梅, 等. 基于污点分析的源代码脆弱性检测技术[J]. 小型微型计算机系统, 2009, 30(1): 78-82.

[24] Reps T, Horwitz S, Sagiv M. Precise interprocedural dataflow analysis via graph reachability [C] // Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1995: 49-61.

[25] Android Developers. Android debug bridge [EB/OL]. (2017-01-11) [2017-01-12]. https://developer.android.com/studio/command-line/adb.html?hl=zh-cn.

[26] Android Developers. Requesting permissions. [EB/OL]. (2016-12-21) [2017-01-12]. https://developer.android.com/guide/topics/permissions/requesting.html.