软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007142]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

状态转换图制导的 ARP 错误检测方法*

林高毅[1], 崔展齐[1], 陈 翔[2], 郑丽伟[1]

[1](北京信息科技大学 计算机学院, 北京 100101)

[2](南通大学 信息科学技术学院, 江苏 南通 226019)

通信作者: 崔展齐, E-mail: czq@bistu.edu.cn

摘 要: Android 应用开发人员需要在保持应用频繁更新的同时快速检测出应用中 Android 运行时权限 (Android runtime permission, ARP) 错误. 现有的 Android 应用自动化测试工具通常未考虑 ARP 机制, 无法有效测试 Android 应用内的权限相关行为. 为帮助开发人员快速检测出应用中 ARP 错误, 提出状态转换图制导的 Android 应用运行时权限错误检测方法. 首先, 对被测应用 APK 文件进行权限误用分析, 插桩 APK 文件中可能导致 ARP 错误的 API, 并对 APK 文件重新签名; 然后, 安装插桩后的 APK 文件, 动态探索应用以生成其状态转换图 (state transition graph, STG); 最后, 使用 STG 制导自动化测试, 快速检测出应用中 ARP 错误. 基于所提出方法实现原型工具 RPBDroid, 并与 ARP 错误动态检测工具 SetDroid、PermDroid 和传统自动化测试工具 APE 进行对比实验. 实验结果表明, RPBDroid 成功检测出 17 个应用中的 15 个 ARP 错误, 比 APE、SetDroid、PermDroid 分别多 14、12 和 14 个. 此外, 相比于测试工具 SetDroid、PermDroid 和 APE, RPBDroid 检测 ARP 错误的平均用时分别减少 86.42%、86.72% 和 86.70%.

关键词: Android 应用; Android 运行时权限错误; 权限误用; 自动化测试工具; 状态转换图

中图法分类号: TP311

中文引用格式: 林高毅, 崔展齐, 陈翔, 郑丽伟. 状态转换图制导的ARP错误检测方法. 软件学报. http://www.jos.org.cn/1000-9825/7142.htm

英文引用格式: Lin GY, Cui ZQ, Chen X, Zheng LW. State Transition Graph Guided Testing Approach for Detecting ARP Bugs. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7142.htm

State Transition Graph Guided Testing Approach for Detecting ARP Bugs

LIN Gao-Yi[1], CUI Zhan-Qi[1], CHEN Xiang[2], ZHENG Li-Wei[1]

[1](School of Computer Science, Beijing Information Science and Technology University, Beijing 100101, China)

[2](School of Information Science and Technology, Nantong University, Nantong 226019, China)

Abstract: While keeping frequent application updates, Android application developers need to detect Android runtime permission (ARP) bugs as quickly as possible. Android applications cannot effectively be tested for permission-related behaviors with automated testing tools since they are rarely designed for ARP bugs. This study proposes a state transition graph guided testing approach for detecting ARP bugs in Android applications. First, it analyzes the APK file of the application under test for permission misuse, instruments the APIs that may cause ARP bugs in the APK file, and re-signs the APK file. Then, it installs the APK file and dynamically explores the application to generate its state transition graph (STG). Finally, it detects ARP bugs quickly by automated testing with the guidance of STG. To evaluate the effectiveness of the approach, the study implements a prototype tool RPBDroid and conducts comparative experiments with the ARP bug detection tools SetDroid, PermDroid, and the automated testing tool APE. The experimental results show that RPBDroid successfully detects 15 ARP bugs out of 17 applications, which detects 14, 12, and 14 more ARP bugs than APE, SetDroid, and PermDroid respectively. In addition, RPBDroid reduces the average time required to detect ARP bugs by 86.42%, 86.72%, and 86.70% in comparison with SetDroid, PermDroid, and APE.

* 基金项目: 江苏省前沿引领技术基础研究专项 (BK202002001); 国家自然科学基金 (61702041); 北京信息科技大学“勤信人才”培育计划 (QXTCP C201906)

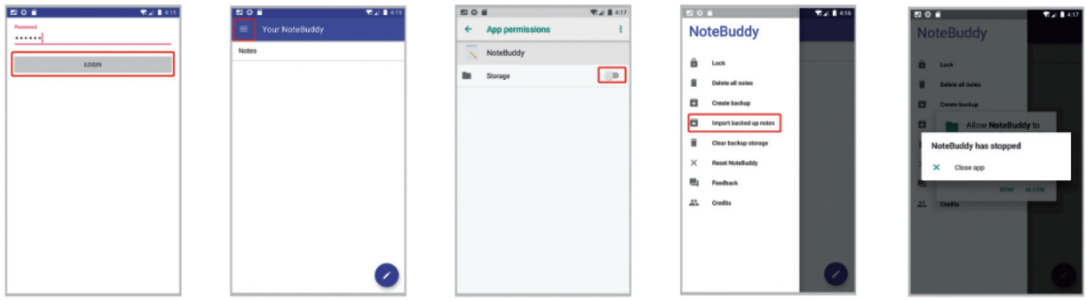
收稿时间: 2023-01-10; 修改时间: 2023-10-29; 采用时间: 2024-01-01; jos 在线出版时间: 2024-04-24

Key words: Android application; Android runtime permission (ARP) bugs; permission misuse; automated testing tools; state transition graph (STG)

为保护用户隐私, Android 平台的权限机制会限制应用访问受限数据 (如用户联系信息) 和执行受限操作 (如录制音频)[1]. 在 Android 6.0 (API 级别 23) 及更高版本中引入了 Android 运行时权限 (Android runtime permission, ARP) 机制后, 应用可以在运行时向用户请求权限, 而用户可以在应用运行时随时撤销或授予权限. ARP 机制的引入有效改善了对数据隐私的保护和应用的使用体验[2], 同时也给 Android 应用的开发和测试带来了新挑战. 为确保应用在任何权限授予情况下都能正常运行, 需要对应用进行更加充分的测试.

Android 平台中受权限保护的 API 数量众多, 此类 API 需被授予所需权限才能使用[3]. 如果 Android 应用在调用受权限保护的 API 前未正确检查或请求相关权限 (本文后续称为权限误用), 可能会导致应用行为异常, 甚至是应用崩溃. 这种 Android 运行时权限问题被称为 ARP 错误[4]. 正如应用权限最佳做法中提到的: “无论使用哪个 API 级别, 用户都可以开启或关闭任何应用的权限. 您应测试以确保您的应用能在各种权限情境中正常运行”[5], ARP 错误会直接影响应用可用性, 在发布应用前应对其充分测试以减少此类错误. Wang 等人[4]的研究发现, 开发人员虽然已经认识到 ARP 机制, 并会使用同步权限管理 (在调用受权限保护 API 前检查权限状态) 和异步权限管理 (在不同的回调函数中异步调用受权限保护 API 和检查权限状态) 模式进行权限检查, 但开发人员难以对应用中所有组件进行检查, 导致应用中仍存在 ARP 错误. 另有研究表明, 开发人员在没有自动化工具支持的情况下很难发现 ARP 错误[6]. 尽管已有大量工作对 Android 自动化测试进行了研究, 但这些传统的自动化测试工具通常未考虑 ARP 机制, 无法有效测试应用的权限相关行为, 以保证 Android 应用在不同权限授予情况下的健壮性.

静态分析方法可用于检测 Android 应用 ARP 错误, 但通常需要开发人员对静态分析的检测结果进行人工确认, 开销较大且效率较低. 以 APER[4]为例, 它提取受权限保护 API 的调用上下文, 并结合数据流及其提出的过程内、过程间等 4 种权限管理模式来定位错误, 比 ARPDroid[7]、revDroid[8]等静态分析方法取得了更高的准确率. 但是, APER 的检测结果中仍存在大量误报, 需通过开发人员手动验证. 面对 Android 应用由大量复杂事件驱动的 GUI 相关行为, 开发人员人工验证错误的开销较大. 例如, 静态工具分析 NoteBuddy (https://www.f-droid.org/en/packages/nl.yoerinijs.notebuddy) 应用后得到的分析报告认为 API “android.os.Environment.getExternalStorage Directory”受权限误用影响, 调用此 API 可能导致 ARP 错误, 但开发人员要成功触发该 ARP 错误需要依次完成图 1(a)–图 1(e) 所示步骤. 此外, 要触发 NoteBuddy 应用内 ARP 错误还需要对应用的权限进行操作, 即图 1(c). 现有的自动化测试工具在测试过程中通常并未考虑应用权限, 因此无法有效检测应用中的 ARP 错误.



(a) Fill in text and touch (b) Touch (c) Revoke permission (d) Touch (e) Crash

图 1 触发 NoteBuddy 应用 ARP 错误操作步骤

如能将静态分析的结果用于制导自动化测试, 可缩小探索范围, 更有针对性地测试 Android 应用中可能触发 ARP 错误的控件, 在测试过程中引入对应用权限的操作, 则可进一步提高检测效率. 为此, 本文提出了一种 Android 应用 ARP 错误动态检测方法, 它通过自动探索被测应用构建状态转换图 (state transition graph, STG), 并结合静态分析结果有针对性地进行测试, 以快速检测应用中的 ARP 错误. 首先, 对被测应用的 APK 文件进行权限误用分析, 获取受权限误用影响的 API, 插桩 APK 文件内受影响的 API, 并对 APK 重新签名. 然后, 将重新签名后的 APK 安装到 Android 设备上, 动态探索被测应用构建 STG, 并在动态探索的过程中记录能够调用受影响 API 的控

件. 最后, 使用 STG 制导自动化探索被测应用, 并输出可触发 ARP 错误的测试脚本. 为评估所提出方法的有效性, 我们基于所提出的状态转换图制导的 Android 应用运行时权限错误检测方法实现了原型工具 RPBDroid (runtime permission bug detector for Android applications), 在 APER[4]中使用的 17 个 Android 应用上进行了实验, 并与考虑到 ARP 机制的动态检测工具 PermDroid[9]、SetDroid[10]和传统的自动化测试工具 APE[11]进行了对比. 实验结果表明, RPBDroid 比 APE、SetDroid、PermDroid 分别多检测到 14、12 和 14 个 ARP 错误. 在检测应用 ARP 错误用时方面, RPBDroid 的平均用时比 APE、SetDroid 和 PermDroid 分别减少了 86.70%、86.42% 和 86.72%.

本文的主要贡献如下.

● 提出了一种状态转换图制导的 Android 应用运行时权限错误检测方法, 通过插装和动态构建 STG 将代码中的 API 调用和 GUI 界面中的控件进行了映射, 结合静态分析定位 Android 应用程序中受权限误用影响的 API, 以有针对性地测试受影响的 API, 从而减少探索应用内的非必要路径, 提升 ARP 错误检测效率.

● 基于所提出方法设计并实现了原型工具 RPBDroid, 在一组真实的 Android 应用上进行了实验, 并与测试工具 APE、SetDroid、PermDroid 进行对比, 以评估 RPBDroid 检测 Android 应用 ARP 错误的有效性和效率.

本文第 1 节通过一个示例介绍本文研究动机. 第 2 节详细描述状态转换图制导的 Android 应用运行时权限错误检测方法. 第 3 节介绍 RPBDroid 检测 Android 应用 ARP 错误的实验结果, 并进行有效性分析. 第 4 节对 RPBDroid 的检测 ARP 错误用时情况、局限性和有效性进行讨论. 第 5 节介绍相关工作. 第 6 节总结全文.

1 研究动机示例

图 2 展示了一个 Android 应用 NoteBuddy 中由于缺失权限检查而导致的 ARP 错误, 该应用可从 F-Droid (https://f-droid.org/en/) 上获取. 通常, 用户在应用的权限授予界面中授予应用存储权限, 然后当用户在 NotesActivity 中执行导入已备份的笔记操作时, 笔记才可被成功导入. 如果用户授予 NoteBuddy 应用相关权限并使其进入后台运行, 执行一些不相关操作 (例如操作其他不相关应用或系统功能), 然后撤销 NoteBuddy 的权限. 在这种情况下, 当用户返回 NoteBuddy 应用并尝试导入已备份的笔记时, 应用会发生崩溃.

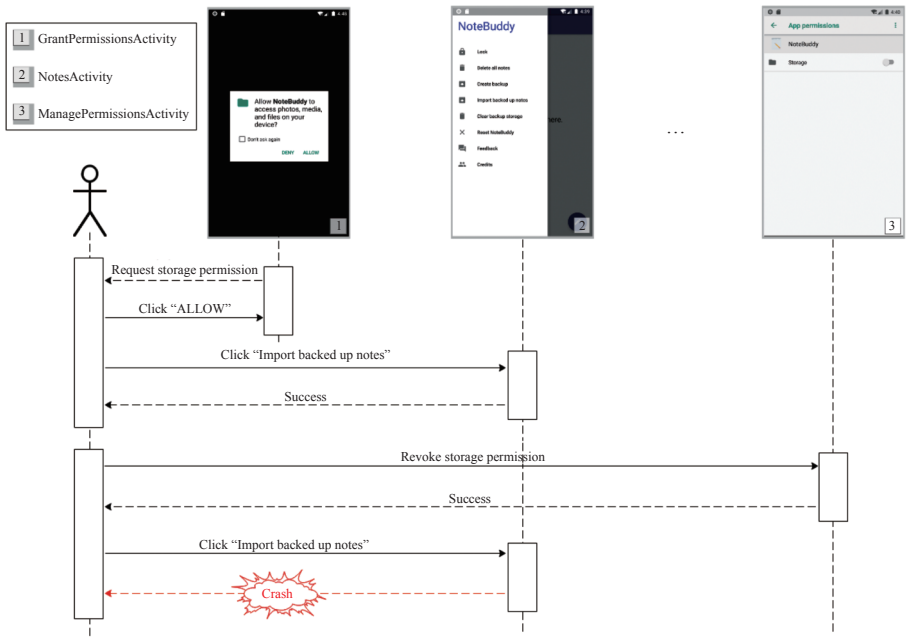


图 2 NoteBuddy 应用中的 ARP 错误

以图 2 所示的 NoteBuddy 应用中的 ARP 错误为例, 使用自动化测试工具 SetDroid[10]、PermDroid[9]和 APE[11]测试 2 h, 上述工具分别执行了 799、2 382、8 251 次操作, 但均不能检测出应用中的 ARP 错误. 分析发现, 面对应

用所包含的大量控件以及应用可能用到的多种危险权限, 上述工具难以准确定位能调用受权限误用影响 API 的控件, 并确定操作该控件前应该撤销的权限. 其中, SetDroid 和 PermDroid 通过使用不同权限组合对 Android 应用进行充分测试, 假设 Android 应用包含了 m 个控件, 使用了 n 个危险权限, 若在不同权限组合的情况下对该应用进行充分测试, 则需要执行 $m\times 2^n$ 次操作[9], 这将消耗大量测试资源, 导致 SetDroid 和 PermDroid 难以在有限的测试时间内检测出应用内的 ARP 错误. 对于通用的 Android 应用测试工具 APE, 在设计过程中并未考虑对应用权限进行操作, 因此难以有效检测 ARP 错误. 此外, 我们还使用静态工具 APER 对 NoteBuddy 应用进行了分析, 尽管可从静态分析结果中获取可能会受权限误用影响 API “android.os.Environment.getExternalStorageDirectory”的调用上下文, 以及该 API 所需的危险权限“android.permission.READ_EXTERNAL_STORAGE”和“android.permission.WRITE_EXTERNAL_STORAG E”, 但开发人员仍然很难利用这些信息设计出能够触发该 ARP 错误的事件序列, 以确认疑似 ARP 错误.

要成功触发该 ARP 错误必须找到能够调用受权限保护 API 的控件, 并在撤销对应权限后操作相应控件. 然而, Android 应用是事件驱动的, 其测试用例通常由 GUI 事件构成, 通过在不同权限组合下遍历操作应用内控件的方式触发 ARP 错误将消耗大量测试资源. 如能利用静态分析获取可能导致 ARP 错误的 API 及其所需权限, 确定该 API 与控件间的调用关系, 并在后续测试过程中将测试资源集中于调用可能导致应用 ARP 错误的控件上, 从而提高测试效率.

本文所提出的状态转换图制导的 Android 应用运行时权限错误检测方法将静态分析与自动化测试相结合, 首先对 NoteBuddy 应用进行静态分析, 可获取受权限误用影响的 API 及其所需危险权限; 然后通过自动化探索 NoteBuddy 应用, 以动态生成 STG, 并定位能调用受影响 API 的控件; 最后使用 STG 制导自动化测试, 以检测 ARP 错误. 使用基于本文方法实现的原型工具 RPBDroid 对 NoteBuddy 应用进行测试, 仅需执行 54 次操作, 花费 160 s 就能检测出该 ARP 错误. 相比于自动化测试工具 SetDroid、PermDroid 和 APE, 有效提升了检测 ARP 错误的效率.

2 状态转换图制导的 Android 应用运行时权限错误检测方法

本文所提出的状态转换图制导的 Android 应用运行时权限错误检测方法框架如图 3 所示. 首先, 对被测应用的 APK 文件进行权限误用分析, 对受权限误用影响的 API 进行插桩, 并对 APK 重新签名; 其次, 将插桩后的 APK 安装到 Android 设备上, 动态探索被测应用以生成 STG, 并记录目标控件集, 即能够调用受影响 API 的控件集; 最后, 使用 STG 制导动态探索被测应用, 识别当前界面中的控件信息, 根据 STG 和目标控件集选择控件进行操作, 并检测应用运行状态, 输出能复现 ARP 错误的测试脚本.

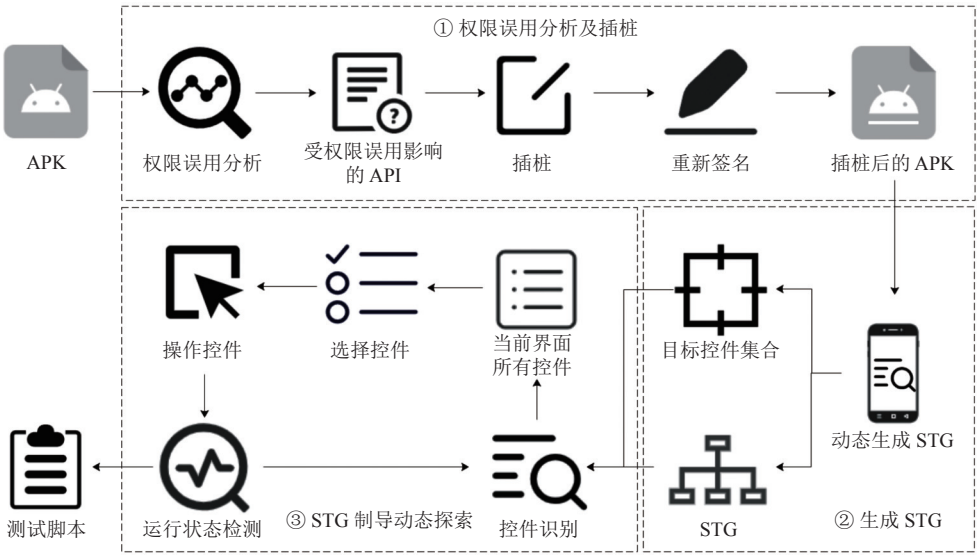


图 3 状态转换图制导的 Android 应用运行时权限错误检测方法框架图

2.1 权限误用分析及插桩

触发 ARP 错误需要在应用程序运行过程中, 从其众多可执行控件中选择并按特定顺序操作调用受权限误用影响的 API 所关联控件, 即操作控件所产生的调用上下文 (calling context) 中包含受权限保护的 API, 但在调用该 API 前未正确检查或请求相关权限. 图 4 为 pyLoad 应用 (https://f-droid.org/en/packages/org.pyload.android.client) 中的代码片段, 在第 25 行处通过调用 API“getExternalStorageDirectory”获取 SD 卡路径, 并在第 30、31 行使用. 该 API 受权限“READ_EXTERNAL_STORAGE”和“WRITE_EXTERNAL_STORAGE”保护, 而 pyLoad 应用在第 25 行调用此 API 时未进行权限检查, 若应用在上述权限未被授予的情况下调用此 API, 会导致应用在运行过程中崩溃.

行号	源代码
24	public final static int CHOOSE_FILE = 0;
25	public final static String SD_CARD = Environment.getExternalStorageDirectory().getPath();
26	private File currentDir;
27	private FileArrayAdapter adapter;
28	public void onCreate(Bundle savedInstanceState) {
29	super.onCreate(savedInstanceState);
30	currentDir = new File(SD_CARD);
31	fill(currentDir);
32	}

图 4 pyLoad 应用内引发 ARP 错误的代码示例

触发 ARP 错误的前提是在应用运行过程中操作控件调用受权限误用影响的 API. Android 应用开发人员通过软件开发工具包 (software development kit, SDK) 中提供的大量 API 与服务和硬件交互, 以实现应用功能[12], 为了触发 ARP 错误, 首先需要定位应用中受权限误用影响的 API. 此外, 面对 Android 应用中包含的大量 GUI 事件, 控件与受影响 API 间的调用关系难以确定.

为解决上述问题, 首先, 对被测应用进行静态权限误用分析, 定位应用内疑似受权限误用影响的 API, 即可疑 API. 在 Wang 等人[4]的工作中, 他们分析了 Android 官方文档中 Android 6.0 版本以后受权限保护的 API, 给出了 API 与权限映射关系列表 (https://github.com/sqlab-sustech/APER-mapping). RPBDroid 同样关注该列表中的 API 与权限映射关系, 使用 APER 进行静态分析以获取受权限误用影响的 API. APER 是一个权限误用检测工具, 它以 APK 文件作为输入, 执行组件内和组件间的静态分析, 以检查应用内受权限保护的 API 被调用前是否会进行权限检查. 若发现某个受权限保护的 API 缺失相关权限检查, APER 则认为该 API 受权限误用影响, 并将其输出. RPBDroid 将 APER 输出的 API 标记为可疑 API.

接下来, 通过插装确定控件与可疑 API 间的调用关系. Soot (https://github.com/soot-oss/soot) 是一种字节码分析和修改工具, 可直接对 APK 文件进行修改, RPBDroid 使用 Soot 进行插装. 具体来说, RPBDroid 在调用可疑 API 的位置后插入 Android SDK 提供的信息级别日志消息输出方法 android.util.Log.i() (例如, 在图 4 第 25 行后插桩), 以在后续动态探索过程中通过监控日志确定控件与可疑 API 的调用关系. 需要注意的是, 对 APK 文件插桩后, 其签名将被破坏, 无法直接安装运行, 需使用签名工具 jarsigner (https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html) 对 APK 重新签名.

2.2 生成 STG

Android 应用是事件驱动的, 由大量界面和可操作 UI 控件组成. 传统 Android 应用自动化测试工具的测试范围通常包括应用的所有可执行路径, 这会在与运行时权限错误无关的路径上浪费大量测试资源, 无法有效生成事件序列以触发 ARP 错误.

为了有针对性地测试可能导致应用 ARP 错误的控件, 需要获取控件与应用界面的所属关系, 以及应用状态间的转换关系. 有限状态机 (finite state machine, FSM) 是一种可以表示有限个状态以及在这些状态之间的转移和动作等行为的工具[13]. Android 应用通过操作控件可实现不同界面间的跳转, 与 FSM 形式相似. 因此, 我们参照刘攀等人[14]和雷斌等人[15]工作中关于 FSM 的描述, 对 STG 进行定义.

在本文中, 应用状态是应用界面的抽象, 使用控件集来表示应用状态, 控件集表示应用当前状态下所有可操作的控件. 当应用中可操作的控件 (包括可操作控件的数量或各控件属性) 发生变化时, 则认为应用状态改变. 例如,

在图 1 中, (a) 界面和 (b) 界面对应于两个应用状态. 控件集可通过分析控件层次树 (widget hierarchy tree) 获取, 树中叶结点即为可操作的控件, 在叶子结点中记录了控件属性信息.

定义 1. STG. Android 应用 A 的 STG 为四元组 $G=(S,\Sigma,\delta,s_0)$. 其中,

- S 是状态集.
- Σ 为输入事件集, 即对应用界面中的控件进行操作 (例如, 点击控件) 集合.
- δ 为状态转移函数, $\delta: S\times\Sigma\to S$, 每次转换的形式为 $\delta(s_u,e_u^a)=s_v$ 或 $\delta(s_u,e_u^b)=s_u$, 前者表示对应用 A 进行 $e_u^a\in\Sigma$ 操作可使其由状态 $s_u\in S$ 转换为 $s_v\in S$, 后者表示进行 $e_u^b\in\Sigma$ 操作不使应用 A 状态 s_u 发生转换.
- $s_0\in S$ 为初始状态, 即应用 A 的初始界面.

对于状态 s_u 下的可操作控件集 $W_u=\{w_u^1,\ldots,w_u^a,\ldots\}$, $Locate(w_u^a)=s_u$, 表示控件 w_u^a 所属的应用状态为 s_u. 对于状态 s_u 和 s_p, 若状态 s_u 至状态 s_p 可达, 则 $Path(s_u,s_p)=<e_u^x,\ldots,e_r^y,\ldots>$, 表示由状态 s_u 转换至状态 s_p 需要进行的操作队列, 其中, $e_r^y=OperateWidget(w_r^y)$ 是在状态 s_r 下对控件 w_r^y 进行的操作; 若状态 s_u 至状态 s_p 不可达, 则 $Path(s_u,s_p)=\varnothing$.

为构建 STG 并定位目标控件, 将第 2.1 节中插桩、签名后得到的 APK 文件安装到 Android 设备上, 使用自动化测试工具对安装后的应用进行探索, 在探索应用的过程中动态构建 STG, 并记录能调用受权限误用影响 API 的控件.

算法 1 描述了构建 STG 和目标控件集合的算法, 输入为插桩后的 APK 文件 f 和最长探索时间 T_{explore}, 输出为 STG G 和目标控件集合 W_{target}. 第 1 行, 初始化 STG G 和目标控件集合 W_{target}. 第 2 行, 安装 APK 文件 f, 启动应用 A, 记录应用初始状态 s_0, 并将状态 s_0 加入状态集 S. 随后, 开始动态探索应用以构建其 STG (第 3–15 行). 第 4–8 行, 根据 STG 获取应用当前状态 s_i, 并分析获取其下可操作的控件集合 W_i, 按控件位置依次选择一个控件 $w_i^m\in W_i$ 进行操作, 并记录操作控件后产生的系统日志 l_i^m. 操作控件后分析当前界面控件集合 W_j (第 10 行), 若 $W_i!=W_j$, 表明应用进入新的状态 s_j, 即 $\delta(s_i,OperateWidget(w_i^m))=s_j$, 则将状态 s_j 加入状态集 S (第 12 行); 反之, 则认为没有发生状态转换, 即 $\delta(s_i,OperateWidget(w_i^m))=s_i$, 不更新状态集 S. 此外, 若在操作控件 w_i^m 后监控到插桩内容, 则将控件 w_i^m 加入目标控件集合 W_{target}, 即操作控件 w_i^m 将会调用可疑 API (第 13, 14 行). 不断重复上述步骤, 达到最长探索时间后结束探索 (第 3 行), 返回 STG G 和目标控件集合 W_{target} (第 15 行).

算法 1. 构建 STG 和目标控件集合.

输入: 插桩后的 APK 文件 f, 最长探索时间 T_{explore};
输出: STG $G=(S,\Sigma,\delta,s_0)$, 目标控件集合 W_{target}.

1. Initialize G, W_{target} /*初始化 STG 和目标控件集合*/
2. A = Install(f), s_0 = Launch(A), S.add(s_0) /*安装并启动应用 A, 记录应用起始状态 s_0, 将 s_0 加入状态集 S */
3. WHILE explore time < T_{explore} do
4. s_i = GetStateOfSTG(G, A) /*根据 STG 获取应用当前状态*/
5. W_i = GetWidgets(s_i) /*获取应用当前状态下的控件集合*/
6. Σ.addEvent(W_i) /*将所有可对应用界面进行的操作加入集合 Σ */
7. $OperateWidget$ (w_i^m = SelectWidget(W_i)) /*从控件集合中选择控件 w_i^m, 操作控件 w_i^m */
8. l_i^m = GetLog(A) /*获取操作控件 w_i^m 后的系统日志*/
9. s_j = GetCurrentState(A) /*获取操作控件后的应用状态*/
10. W_j = GetWidgets(s_j)
11. IF $W_i!=W_j$ /*若操作控件后, 应用状态改变, 则扩展 G */
12. S.add(s_j)
13. IF l_i^m contains instrument contents /*若系统日志中包含插装信息, 则控件 w_i^m 为目标控件*/
14. W_{target}.add(w_i^m)
15. RETURN G, W_{target}

图 5 为 NoteBuddy 应用的部分 STG 示例. 图中节点表示应用状态, 应用状态由控件集表示, 例如, 图 5 中状态 s_u 的控件集为 $\{w_u^a, w_u^c\}$. 实线箭头表示状态转换, 应用的状态转换由操作控件所触发, 例如, 图 5 中操作控件 w_u^a 可使应用状态由 s_u 转换至 s_v.

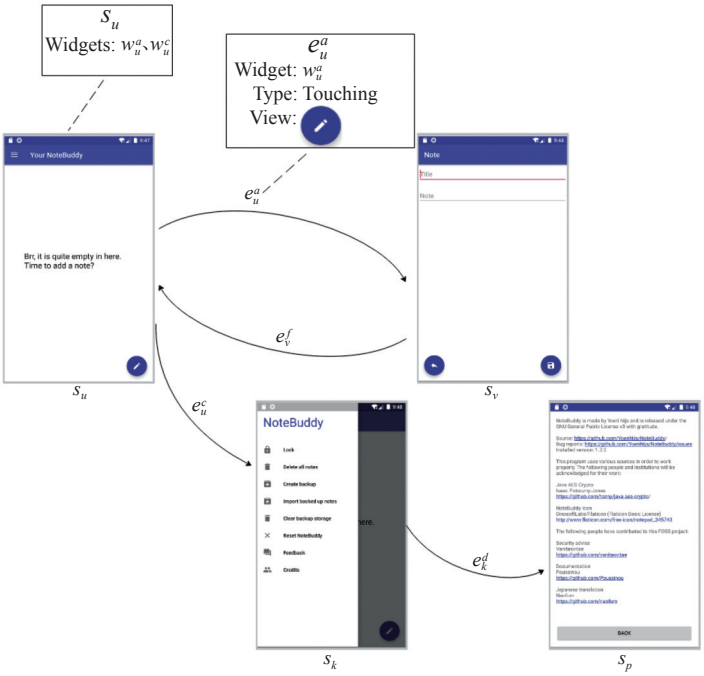


图 5 NoteBuddy 应用的部分 STG

2.3 STG 制导动态探索

获取被测应用 STG 和目标控件集合后, 可将其用于制导自动化测试, 并在探索过程中引入对应用权限的操作, 以快速检测被测应用的 ARP 错误. 需要注意的是, 部分开发人员开发应用时会意识到 ARP 机制, 并通过同步权限管理和异步权限管理模式对应用权限进行管理. 但是, 开发人员难以检查应用中所有路径, 应用中仍可能存在 ARP 错误. 需要设计有效的探索策略, 以对上述两种场景进行测试.

为此, 本文设计了两阶段的探索策略: 首先根据 STG 和目标控件集合快速到达目标控件所在界面后, 检查并撤销应用相关权限, 对目标控件进行操作, 以完成对同步权限使用场景的测试; 接下来, 以目标控件所在界面为起点, 继续探索应用, 期间引入对应用权限的操作, 以完成对异步权限使用场景的测试.

算法 2 描述了 STG 制导动态探索的算法, 输入为 STG G、目标控件集合 W_{target}、被测应用 A 和最长测试时间 T_{test}, 输出为可触发 ARP 错误的测试脚本 R. 其中, 测试脚本 R 记录探索过程中操作控件和撤销应用权限时所使用的命令, 以便开发人员后续利用该脚本复现 ARP 错误. 第 1 行, 初始化测试脚本 R. 随后, 对于目标控件 $w_p^q \in W_{\text{target}}$, 分阶段使用不同策略选择控件进行操作 (第 2–24 行). 第 1 阶段 (第 3–15 行) 对同步权限使用场景进行测试, 启动应用 A (第 3 行), 获取控件 w_p^q 所属应用状态 $s_p \in S$ (第 4 行). 根据 STG 获取状态 s_0 至状态 s_p 需要进行的操作队列 $Path(s_0, s_p) =< e_0^t, \ldots, e_r^y, \ldots >$ (第 5 行), 并根据该队列操作应用中的控件, 使应用状态转换为 s_p (第 6–9 行). 到达状态 s_p 后, 首先分析可疑 API 所需权限的授予情况, 若相关权限已被授予, 则在撤销权限后操作目标控件 w_p^q; 反之, 则直接操作控件 w_p^q (第 10–13 行). 第 2 阶段 (第 16–23 行) 对异步权限使用场景进行测试, 从应用当前状态开始基于深度优先遍历策略选择并操作控件 (第 19 行), 即每次从应用当前界面中选择一个控件操作, 当执行一条事件序列后, 若无法探索到更多状态时, 则返回状态 s_p, 并探索以 s_p 为起点的其他路径. 在每次操作控件前检查可疑 API 所需权限的授予情况, 若权限已被授予, 则撤销此权限以确保在未授予权限的情况下操作控件

(第 17 行). STG 中以状态 s_p 为起点的路径被充分探索后 (第 16 行), 关闭应用 (第 23 行), 从目标控件集合 W_{target} 选择其他控件并重复上述两阶段 (第 2 行). 在探索过程中, 使用测试脚本 R 记录对控件和权限进行的操作 (第 9、11、13、18、20 行). 发现 ARP 错误后, 停止探索并返回测试脚本 R (第 14, 15, 21, 22 行). 若在最长测试时间 T_{test} 内未发现 ARP 错误 (第 2 行), 则停止探索, 不生成测试脚本 (第 24 行).

算法 2. STG 制导动态探索.

输入: STG $G=(S,\Sigma,\delta,s_0)$, 目标控件集合 W_{target}, 被测应用 A, 最长测试时间 T_{test};
输出: 测试脚本 R.

1. Initialize R /*初始化测试脚本*/
2. FOR each widget $w_p^q \in W_{\text{target}}$ and test time < T_{test} do
3. Launch(A) /*启动被测应用 A */
4. $s_p \leftarrow Locate(w_p^q)$
5. $list = Path(s_0, s_p)$ /*获取从状态 s_u 到 s_p 所需进行的操作队列*/
6. WHILE $list$! = NULL do
7. $e_r^y = list$. dequeue() /*取出操作队列中的队首操作 e_r^y */
8. do(e_r^y) /*进行 e_r^y 操作*/
9. R.add(e_r^y) /*将对应用控件进行的操作记录至脚本 R */
10. CheckAndRevokePerm(A) /*检查并撤销应用权限*/
11. R.add(CheckAndRevokePerm(A)) /*将对应用权限进行的操作保存至测试脚本 R */
12. $OperateWidget(w_p^q)$ /*操作目标控件 w_p^q */
13. R.add($OperateWidget(w_p^q)$)
14. IF trigger ARP bug
15. RETURN R
16. WHILE path is not fully explored do /*循环探索被测应用, 直至 G 中以 S_p 为起点的路径被充分探索*/
17. CheckAndRevokePerm(A)
18. R.add(CheckAndRevokePerm(A))
19. $OperateWidget$(DFSBasedSelect(G, A)) /*基于深度优先遍历策略选择控件并进行操作*/
20. R.add($OperateWidget$(DFSBasedSelect(G, A)))
21. IF trigger ARP bug
22. RETURN R
23. Kill(A) /*关闭应用 A */
24. RETURN NULL

3 实验及评估

3.1 实验设计

基于所提出方法, 我们在 DroidBot[16]框架的基础上实现了自动化测试工具 RPBDroid, 它以被测应用中受权限误用影响的 API 为目标, 自动化地生成能够触发 ARP 错误的事件序列. DroidBot 是一个轻量级的测试输入生成工具, 不需要插桩就能在大多数设备上与 Android 应用交互, 并允许用户自定义分析应用界面的算法和选择控件的策略. Nighthawk[17]、MobiPurpose[18]、OwlEye[19]、NoMoATS[20]等工具都是将 DroidBot 作为基础框架实现的. 为定位可疑 API, RPBDroid 使用 APER[4]进行权限误用静态分析. APER 以 APK 文件为输入, 使用应用内控件产生

的 API 调用上下文和数据流定位并输出受权限误用影响的 API. 为生成应用 STG, 使用自动化测试工具 DroidBot 中基于模型的探索策略动态探索应用, 以在探索应用的过程中构建 STG, 并确定受权限误用影响的 API 与控件之间的调用关系.

实验的开发和运行环境为: 16 GB 内存, AMD® Ryzen 5 5600h 处理器, Ubuntu 20.04, Android SDK (6.0–12.0), JDK 1.8.0, Python 3.8.10.

为评估 RPBDroid 的有效性, 提出以下 3 个研究问题.

● RQ1: 与其他 Android 应用自动化测试工具相比, RPBDroid 检测 ARP 错误的有效性和效率如何?

RPBDroid 旨在利用静态分析结果提高检测 ARP 错误的效率, 与其他 Android 应用运行时权限错误检测工具以及通用 Android 应用自动化测试工具相比, RPBDroid 的有效性和效率如何?

● RQ2: STG 制导如何影响 RPBDroid 的性能?

RPBDroid 通过 STG 制导以快速到达目标控件所在界面, STG 制导如何影响 RPBDroid 的性能?

● RQ3: 对应用权限的操作如何影响 RPBDroid 的性能?

考虑到 Android 运行时权限机制, RPBDroid 在检测应用 ARP 错误时会对应用权限进行操作, 这是否会影响 RPBDroid 的性能?

在实验过程中, 我们沿用了 Zhao 等人[21]的实验设置, 将每个工具的最长测试时间设置为 2 h. 为更准确地获取相关测试工具检测 ARP 错误的用时, 实验使用各测试工具在实验对象上分别运行 10 次, 并将实验结果取平均值后记为最终用时. 此外, 若相关测试工具在最长测试时间内未检测到指定的 ARP 错误, 则在计算检测 ARP 错误的时间时记为 2 h.

3.2 实验对象

为了对 RPBDroid 的有效性进行评估, 我们以 Wang 等人[4]的研究对象为基础, 选取本文的实验对象. 在 Wang 等人的工作中, 他们从移动应用市场 F-Droid 中收集了 214 个开源 Android 应用, 使用 APER 分析并手动验证后, 确认其中有 23 个由于缺少权限检查而导致 ARP 错误的 Android 应用. 本文对上述 23 个 Android 应用进行了分析, 具体来说, 我们利用 Wang 等人工作中提供的应用名和包名在 F-Droid 检索相关应用, 除已下架的 1 个应用外, 获得 22 个 Android 应用. 接下来, 我们尝试复现上述 Android 应用中的 ARP 错误, 去除 5 个错误无法被复现的应用后, 将剩余的 17 个 Android 应用全部作为本文实验对象. 每个应用中都含有 1 个可导致应用崩溃的 ARP 错误. 表 1 展示了本文实验对象的基本信息, 可以看到, 本文实验对象包含了 6 种类别的应用, 平均代码行数为 25.0k. 触发本文实验对象中 ARP 错误所需最大步骤数为 8, 最小步骤数为 3, 平均步骤数为 5. 这进一步说明了有必要通过自动化测试生成相关事件序列, 以对 ARP 错误进行验证. 此外, 对于 17 个作为实验对象的 ARP 错误, 已有 15 个被提交到应用各自的问题跟踪系统, 其中 2 个已被修复, 5 个已经得到了开发人员确认. 另有 2 个 ARP 错误由于应用的错误跟踪系统已经关闭, 因此未被提交.

表 1　实验对象基本信息

实验对象	类别	版本	代码行	步骤数	类型	状态	F-Droid链接
pyLoad	Tools	0.3.7	89.8k	7	crash	fixed	https://f-droid.org/en/packages/org.pyload.android.client
LibreBusTO	Productivity	1.14	91.6k	3	crash	fixed	https://www.f-droid.org/en/packages/it.reyboz.bustorino
EtchDroid	Tools	1.5	12.1k	4	crash	confirmed	https://www.f-droid.org/en/packages/eu.depau.etchdroid
MemeTastic	Entertainment	1.6.7	24.8k	5	crash	confirmed	https://www.f-droid.org/en/packages/io.github.gsantner.memetastic
Meshenger	Productivity	3.0.3	17.7k	5	crash	confirmed	https://www.f-droid.org/en/packages/d.d.meshenger
MyPosition	Travel & Local	1.3.3	3.6k	4	crash	confirmed	https://www.f-droid.org/en/packages/net.mypapit.mobile.myposition
TopoSuite	Productivity	1.2.0	60.1k	6	crash	confirmed	https://www.f-droid.org/en/packages/ch.hgdev.toposuite
LeafPicGallery	Photography	0.6	26.8k	3	crash	open	https://www.f-droid.org/en/packages/org.horaapps.leafpic
AsteroidOSSync	Android Wear	0.21	21.6k	3	crash	open	https://www.f-droid.org/en/packages/org.asteroidos.sync

表 1 实验对象基本信息 (续)

实验对象	类别	版本	代码行	步骤数	类型	状态	F-Droid链接
Syncopoli	Tools	0.6	4.9k	6	crash	open	https://www.f-droid.org/en/packages/org.amoradi.syncopoli
OINotepad	Productivity	1.5.4	26.8k	5	crash	open	https://www.f-droid.org/en/packages/org.openintents.notepad
SpaRSSDecSync	Tools	1.15.2	29.3k	5	crash	open	https://www.f-droid.org/en/packages/org.decsync.sparss.floss
NoteCryptPro	Productivity	1.44	7.7k	8	crash	open	https://www.f-droid.org/en/packages/com.notecryptpro
OneTimePass	Tools	1.2.2	14.0k	7	crash	open	https://www.f-droid.org/en/packages/com.github.onetimepass
PasseGares	Travel & Local	1.4.1	80.7k	3	crash	open	https://www.f-droid.org/en/packages/fr.nocle.passegares
MinetestMods	Tools	1.9.1	11.1k	3	crash	—	https://www.f-droid.org/en/packages/com.rubenwardy.minetestmodmanager
NoteBuddy	Tools	1.3.2	42.5k	5	crash	—	https://www.f-droid.org/en/packages/nl.yoerinijs.notebuddy

3.3 实验结果分析

3.3.1 RQ1 结果

在本研究问题中, 我们将 RPBDroid 和开源工具 PermDroid[9]、SetDroid[10]、APE[11]就检测 ARP 错误的情况进行对比, 评价标准是能否在有限的时间内快速检测出应用中的 ARP 错误. 其中, PermDroid 被用于检测 Android 应用内 ARP 错误, 它实现了局部穷举的权限组合策略, 以有效地测试权限相关行为, 能在更短的时间内达到更高的权限相关行为代码覆盖率. SetDroid 是一种检测 Android 应用设置缺陷的自动化测试技术, 通过向事件序列中插入设置更改事件, 它能够检测到其他自动化测试工具无法检测到的错误. SetDroid 支持的设置更改事件包含了权限的授予或撤销, 因此同样可用于检测 Android 应用内 ARP 错误. APE 是通用 Android 应用自动化测试工具, 它实现了基于模型的探索策略, 在测试期间利用运行时信息动态优化模型, 相比于 Monkey[22]、Sapienz[23]等工具, APE 能够探索到更多界面, 检测出更多错误.

表 2 为 RPBDroid 和自动化测试工具 SetDroid、PermDroid、APE 检测表 1 中 17 个实验对象中 ARP 错误的情况, 表 2 中“T/O”表示 2 h 内未能检测到相应 ARP 错误. 可以看到, 在指定测试时间内, RPBDroid 可以检测出 15 个 ARP 错误, SetDroid 能够检测出 3 个 ARP 错误, PermDroid 和 APE 均只能检测出 1 个 ARP 错误. 图 6 为 RPBDroid、SetDroid、PermDroid 和 APE 检测 ARP 错误 Venn 图. 从图 6 中可以看出, SetDroid、PermDroid、APE 检测到的 ARP 错误均能被 RPBDroid 检测到.

表 2 RPBDroid、SetDroid、PermDroid、APE 检测 ARP 错误的情况对比 (s)

实验对象	RPBDroid	SetDroid	PermDroid	APE
pyLoad	16	5 952	T/O	505
LibreBusTO	16	T/O	T/O	T/O
EtchDroid	20	T/O	T/O	T/O
MemeTastic	46	T/O	T/O	T/O
Meshenger	91	T/O	T/O	T/O
MyPosition	143	T/O	T/O	T/O
TopoSuite	86	T/O	T/O	T/O
LeafPicGallery	52	T/O	T/O	T/O
AsteroidOSSync	T/O	T/O	T/O	T/O
Syncopoli	T/O	T/O	T/O	T/O
OINotepad	31	T/O	682	T/O
SpaRSSDecSync	78	1 214	T/O	T/O
NoteCryptPro	126	T/O	T/O	T/O
OneTimePass	84	T/O	T/O	T/O
PasseGares	22	T/O	T/O	T/O
MinetestMods	12	5 295	T/O	T/O
NoteBuddy	160	T/O	T/O	T/O
平均用时	905	6 662	6 816	6 806

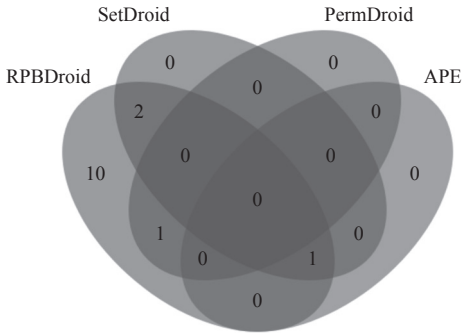


图 6　RPBDroid、SetDroid、PermDroid 和 APE 检测 ARP 错误 Venn 图

对 RPBDroid 无法检测出的 2 个 ARP 错误进行分析, 发现实验对象 Syncopoli 中的 ARP 错误需要在同一个文本框中连续 2 次输入相同内容才能触发, 实验对象 AsteroidOSSync 中的 ARP 错误需要连续 2 次执行权限的授予操作才能触发. 上述 2 个 ARP 错误的触发条件较为苛刻, 而 RPBDroid 的探索策略均难以生成上述操作序列. SetDroid、PermDroid 和 APE 同样未检测到上述 2 个 ARP 错误.

对于 SetDroid、PermDroid 和 APE 检测出的 ARP 错误, 均能被 RPBDroid 检测到, 且 RPBDroid 的用时更少. RPBDroid 比 SetDroid 检测 ARP 错误的平均用时减少了 86.42%, 比 PermDroid 的平均用时减少了 86.72%, 比 APE 的平均用时减少了 86.70%. 这是因为 SetDroid 和 PermDroid 的探索范围是被测应用内所有可达路径, 未考虑目标控件, 浪费了大量测试资源, 其检测 ARP 错误的效率较低. 对于自动化测试工具 APE, 它在设计时并未考虑 Android 运行时权限机制, 在测试过程中无法对应用权限进行操作, 因此无法有效检测 Android 应用中的 ARP 错误, 仅发现了实验对象 pyLoad 中的 1 个 ARP 错误. 这是因为触发实验对象 pyLoad 中的 ARP 错误较为容易, 在调用受权限保护的 API 时未进行权限检查或请求, 因此在未授予相关的情况下直接操作能够调用受保护 API 的控件后即可触发此 ARP 错误.

针对 RQ1 的结论: RPBDroid 检测 Android 应用中 ARP 错误的能力更强. 相比于自动化测试工具 SetDroid、PermDroid 和 APE, RPBDroid 分别多检测到 12、14、14 个 ARP 错误, 检测 ARP 错误的平均用时分别减少了 86.42%、86.72%、86.70%.

3.3.2　RQ2 结果

在本研究问题中, 将与未使用 STG 制导的 RPBDroid 进行对比, 记为 RPBDroid w/o STG. 由于未使用 STG 制导, RPBDroid w/o STG 无法获取应用不同状态间的转换关系, 因此无法有效制导测试到达目标控件所在界面. RPBDroid w/o STG 从被测应用的启动界面开始将根据控件位置依次选择一个控件操作. 由于无法定位目标控件, RPBDroid w/o STG 在每次操作控件前都会检查可疑 API 所需权限的授予情况, 若危险权限已被授予, 则在撤销此危险权限后再操作控件.

表 3 展示了 RPBDroid 和 RPBDroid w/o STG 测试 17 个实验对象的用时情况, 表中“T/O”表示在 2 h 内未能检测出相应 ARP 错误. 第 2、4 列为从实验开始到首次操作目标控件的用时 T_{first} . 第 3、5 列为检测到应用内 ARP 错误的用时 T_{detect} . 可以看到, 在首次操作目标控件的用时方面, RPBDroid 的平均用时比 RPBDroid w/o STG 减少了 72.86%. 在检测应用内 ARP 错误方面, RPBDroid 可以检测出 15 个实验对象中的 ARP 错误, RPBDroid w/o STG 可以检测出 13 个 ARP 错误. 相比于 RPBDroid, RPBDroid w/o STG 少检测到 NoteCryptPro 和 OneTimePass 中的 2 个 ARP 错误. 分析原因后发现, 要到达上述两个实验对象中导致 ARP 错误的界面需要执行较长事件序列, 而未使用 STG 制导的 RPBDroid w/o STG 无法在有限时间内到达目标界面, 因此无法检测相应错误. 对于 RPBDroid 未检测出的 2 个 ARP 错误, RPBDroid w/o STG 同样无法检测到. 对于二者都能检测出的 ARP 错误, 除实验对象 SpaRSSDecSync 和 MinetestMods 内 ARP 错误因均可在应用主界面上直接触发而导致用时相同外, RPBDroid w/o

STG 在其他实验对象上的用时均长于 RPBDroid. RPBDroid 比 RPBDroid w/o STG 检测 ARP 错误的平均用时减少了 51.84%. 这是由于 RPBDroid w/o STG 未使用 STG 制导, 无法快速定位目标控件, 消耗了大量测试资源. 相比之下, 使用了 STG 制导的 RPBDroid 可快速到达目标控件所在界面, 对调用受权限误用影响 API 的控件进行权限检查和撤销操作, 因此其检测 ARP 错误的效率高于 RPBDroid w/o STG.

表 3 RPBDroid 与 RPBDroid w/o STG 测试用时对比 (s)

实验对象	RPBDroid		RPBDroid w/o STG	
	T_{first}	T_{detect}	T_{first}	T_{detect}
pyLoad	9	16	71	95
LibreBusTO	4	16	4	29
EtchDroid	9	20	128	178
MemeTastic	6	46	8	80
Meshenger	17	91	123	186
MyPosition	4	143	4	530
TopoSuite	4	86	4	652
LeafPicGallery	6	52	8	57
AsteroidOSSync	206	T/O	206	T/O
Syncopoli	4	T/O	5	T/O
OINotepad	13	31	64	261
SpaRSSDecSync	3	78	7	78
NoteCryptPro	4	126	4	T/O
OneTimePass	15	84	303	T/O
PasseGares	9	22	74	646
MinetestMods	3	12	5	12
NoteBuddy	13	160	167	345
平均用时	19	905	70	1 879

针对 RQ2 的结论: STG 可帮助 RPBDroid 快速定位目标控件并提升检测 ARP 错误的能力, 相比未使用 STG 制导的 RPBDroid w/o STG, RPBDroid 首次操作目标控件的平均用时减少了 72.86%, 多检测到 2 个 ARP 错误, 且检测 ARP 错误的平均用时减少了 51.84%.

3.3.3 RQ3 结果

在本研究问题中, 将与不对应用权限进行检查和撤销操作的 RPBDroid 进行对比, 记为 RPBDroid w/o perm. RPBDroid w/o perm 在应用运行过程中不对权限进行操作, 其探索策略为从目标控件所属界面执行深度优先遍历策略.

表 4 为 RPBDroid 和 RPBDroid w/o perm 检测 17 个实验对象中 ARP 错误的用时情况, 表中“T/O”表示 2 h 内未能检测出相应 ARP 错误. 可以看到, RPBDroid w/o perm 仅能检测到 2 个实验对象中的 ARP 错误, 而 RPBDroid 能检测出 15 个 (包含 RPBDroid w/o perm 检测到的 2 个错误), RPBDroid 比 RPBDroid w/o perm 检测 ARP 错误的平均用时减少了 85.76%. 这是因为 RPBDroid w/o perm 虽能利用 STG 定位目标控件, 但它没有考虑到 Android 运行时权限机制的特性, 仅进行自动化探索而不对权限做任何操作. 相比之下, RPBDroid 考虑到 Android 运行时权限机制, 在探索过程中引入了对应用权限的操作, 使其检测 ARP 错误的效率显著高于 RPBDroid w/o perm. 分析 RPBDroid w/o perm 仅能检测到的 2 个 ARP 错误后发现, 实验对象 pyLoad 和 OINotepad 在调用受权限保护 API 时未进行任何权限检查和请求操作, 在未授予权限的情况下直接操作目标控件即可触发 ARP 错误.

针对 RQ3 的结论: 对应用权限进行操作可大幅提高 RPBDroid 检测 ARP 错误的效率, 相比于不对应用权限进行操作的 RPBDroid w/o perm, RPBDroid 可多检测到 13 个 ARP 错误, 且检测 ARP 错误的平均用时减少了 85.76%.

表 4　RPBDroid 与 RPBDroid w/o perm 检测 ARP 错误的情况对比 (s)

实验对象	RPBDroid	RPBDroid w/o perm
pyLoad	16	16
LibreBusTO	16	T/O
EtchDroid	20	T/O
MemeTastic	46	T/O
Meshenger	91	T/O
MyPosition	143	T/O
TopoSuite	86	T/O
LeafPicGallery	52	T/O
AsteroidOSSync	T/O	T/O
Syncopoli	T/O	T/O
OINotepad	31	31
SpaRSSDecSync	78	T/O
NoteCryptPro	126	T/O
OneTimePass	84	T/O
PasseGares	22	T/O
MinetestMods	12	T/O
NoteBuddy	160	T/O
平均用时	905	6356

4　讨　论

4.1　检测 ARP 错误用时分析

为进一步了解 RPBDroid 在检测各个实验对象中 ARP 错误时的性能表现, 接下来对其用时分布情况进行讨论, 如图 7 所示. 可以看到, 箱线图上下四分位数的范围较小, 且未出现异常值, 这表明 RPBDroid 检测 ARP 错误的用时在这些实验对象上较稳定, 鲁棒性较好. 分析 RPBDroid 测试过程发现, 这主要得益于利用静态分析结果制导, 能将测试资源优先集中到存在 ARP 错误的路径上, 有效减少对非必要路径的探索.

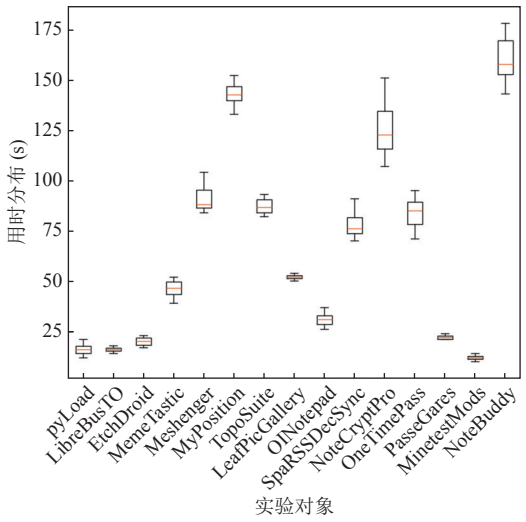


图 7　RPBDroid 检测 ARP 错误用时箱线图

4.2　局限性

RPBDroid 关注的是由于缺少权限检查而引起的 ARP 错误. 如表 1 中各实验对象的 ARP 错误类型所示, 缺少

权限检查而导致的 ARP 错误一般会导致应用崩溃. RPBDroid 目前仅支持检测应用崩溃, 能够检测的错误类型有限. 除因缺少权限检查而导致 ARP 错误外, 应用中还存在另一种由于使用不兼容的权限而引起的 ARP 错误, 这会导致应用行为异常. 受限于当前的测试预言的局限性, RPBDroid 暂不支持检测该类错误. 在后续改进过程中, 可扩展 RPBDroid 的测试预言 (例如, 比较被测对象在不同版本 Android 系统上的执行结果) 使其能够检测到更多类型的 ARP 错误.

4.3 有效性分析

4.3.1 外部有效性

外部有效性威胁一方面来自所选实验对象的代表性, 另一方面来自 RPBDroid 的通用性. 为确保实验对象的代表性, 所选 Android 应用均为来自开源移动应用市场 F-Droid 的真实应用, 涵盖了 6 种应用类别, 平均 25.0k 行代码, 其源代码在 GitHub 或 F-Droid 上开源, 并已被用于相关 Android 应用测试工作. 此外, 各实验对象中的 ARP 错误均是真实存在的, 已被提交到应用各自的错误跟踪系统中, 部分错误已经得到开发人员的确认. 为提高 RPBDroid 的通用性, 我们基于 DroidBot 实现了 RPBDroid. DroidBot 被很多工作所使用, 如 Nightawk[17]、MobiPurpose[18]、OwlEye[19]、NoMoATS[20]等工具都是基于 DroidBot 实现的. 得益于 DroidBot 的轻量化、通用性强的优势, RPBDroid 无需对被测应用进行额外处理就能对其进行测试, 并能在几乎所有的 Android 系统上对不同的 Android 应用进行测试, 以发现其中的 ARP 错误.

4.3.2 内部有效性

内部有效性威胁主要来自定位被测应用内受权限误用影响 API 的准确性和 RPBDroid 探索被测应用过程的正确性. 为了提高定位受权限误用影响 API 的准确性, 本文使用 APER[4]检测被测应用 APK 文件, 分析得到应用内受影响的 API. APER 通过分析多种权限管理模式来定位错误, 其检测权限误用性能优于 ARPDroid[7]、revDroid[8]等其他静态分析工具. 为了确保 RPBDroid 能正确地探索被测应用, 本文使用 DroidBot[16]来驱动 RPBDroid 探索被测应用. 在相关工作中, DroidBot 已被广泛应用于 GUI 测试、兼容性测试、恶意软件分析等领域, 其正确性已得到大量实验的验证. 此外, 本文还对编写的 STG 生成及 STG 制导动态探索功能代码进行了多次检查和测试, 以尽量降低内部有效性威胁.

5 相关工作

本节将介绍相关的 Android 应用 GUI 测试和 ARP 错误检测方法.

5.1 Android 应用 GUI 测试

在 Android 测试领域, 大部分工作集中在 Android 应用 GUI 测试, 这些工作本质上都在试图自动探索被测应用中所有可能的事件组合[24]. 它们实现了不同的 GUI 探索策略 (如: 随机、基于模型的、基于强化学习的探索策略), 以检测应用错误或达到较高的应用测试覆盖率.

部分相关工作使用了随机探索策略, 生成伪随机事件以对被测应用进行模糊测试. Monkey[22]是最常用的基于随机策略的自动化测试工具, 它将 Android 应用的界面视作黑盒, 通过不断与屏幕坐标随机交互以生成 UI 事件, 直至生成的 UI 事件数量达到某一阈值. 尽管随机策略在一些基准应用上取得了较好的效果, 但其生成的测试用例中包含了大量无效或冗余事件, 这将影响测试效率. Dynodroid[25]改进了与屏幕坐标随机交互的探索方式, 它对应用的探索是一个观察、选择、执行的循环. 在观察阶段, Dynodroid 确定控件在当前屏幕上的布局以及每个控件期望的输入类型. 在选择阶段, Dynodroid 会倾向于选择不被频繁选择的控件. 最后, 在执行阶段, Dynodroid 操作所选控件. 此外, Dynodroid 还允许用户暂停自动探索, 可以让用户进行手动操作 (例如: 身份验证、输入密码等) 后恢复探索. EHBDroid[26]则实现了一个新颖的探索策略, 相比于其他 GUI 测试方法, 它不与应用界面交互生成事件, 而是直接调用处理事件的回调方法. 通过这种方式, EHBDroid 可以有效地生成大量传统 GUI 测试方法难以生成的事件.

另一类工作基于静态或动态构建的应用模型生成测试用例. 以 Stoat[27]为例, 它对应用的测试分为两个阶段:

首先使用由加权 UI 探索策略和静态分析增强的动态分析技术来建模被测应用的 GUI 交互模型; 然后利用上一阶段中的模型, 通过迭代的方式指导测试用例生成, 以产生具有高代码覆盖率的事件序列. GUIRipper[28]同样实现了基于模型的探索策略, 被测应用的模型会在探索过程中会动态构建. GUIRipper 执行 DFS 策略, 若当前路径无法探索到被测应用新界面, 则会重新其他路径探索被测应用. GOALEXPLORER[29]构建被测应用模型, 并使用此模型制导动态探索, 因此同样是一种基于模型探索策略的测试工具. GOALEXPLORER 的核心思想是首先对应用的界面和这些界面之间的转换进行静态建模, 生成屏幕转换图 (screen transition graph). 然后, 使用屏幕转换图将应用的动态探索引导到特定的目标. NaviDroid[30]结合静态分析和动态探索提取应用不同状态间的转换关系, 并设计了一种基于上下文的状态合并方法, 以合并相似状态. 此外, NaviDroid 利用动态规划算法遍历所有探索路径, 以覆盖应用所有状态, 并减少重复的探索步骤. 与使用静态 GUI 模型制导测试的方法不同, APE[11]使用动态模型来驱动测试. 在测试期间, APE 会基于观察到的运行时信息逐步细化模型, 以有效平衡模型的大小和精度, 进而提高了 APE 的测试效率. Azim 等人[31]在 Android 应用字节码上进行数据流分析, 以构建高级控制流图来获取活动 (即应用界面) 之间的转换关系, 并用于制导测试, 以快速前往正常情况下难以到达的活动. Lv 等人[32]认为测试过程中获得的事件-活动转换知识对于指导后续测试是有价值的. 他们使用概率模型记录这类知识, 并设计了一个基于概率模型的测试工具 Fastbot2. Fastbot2 利用概率模型引导 GUI 探索, 使其选择并执行可能增加活动覆盖率的事件, 并利用探索过程中获取的信息更新概率模型, 循环执行这一过程直至达到最大测试时间.

还有一类工作利用强化学习技术制导 Android 应用测试. 例如, QBE[33]将强化学习技术用于探索被测应用, 实现了一种完全自动化的黑盒测试方法. QBE 使用强化学习技术生成 Q-Matrix, 以评估操作各控件的概率. 在探索被测应用期间, 基于 Q-Matrix 从应用界面中选择将要操作的控件. Q-testing[34]同样将强化学习技术应用于自动化测试过程, 提出了好奇心驱动的探索策略. Q-testing 在探索过程中会创建并维护一个记录访问过的被测应用状态的集合, 并根据应用当前状态与集合中状态之间的差异来计算强化学习奖励. Q-testing 可以根据被测应用状态重要性的变化, 动态调整特定事件的奖励. 此外, 为了准确划分应用的不同状态, 作者还提出了一种神经网络. Vuong 等人[35]结合随机策略和模型策略的优点提出了一种名为 Q-learning 的强化学习算法, 该算法通过与被测应用交互, 逐步构建被测应用的行为模型并基于此模型生成测试用例, 以尽可能多地探索被测应用功能. ARES[36]是一种基于深度强化学习的方法, 用于自动化黑盒测试 Android 应用. ARES 利用深度神经网络来学习最佳的探索策略, 以提高测试效率. 它具有较强的可扩展性和适用性, 能够处理复杂的应用行为. MARTesting[37]是一种轻量级的基于深度强化学习和多属性融合的 Android 应用自动化测试方法, 它比较应用当前状态与上一状态属性集的差异来移除无效控件, 并将界面上所有控件的属性抽象为神经网络的输入状态, 以解决已有工作中存在的内存消耗大以及基于图像输入的单一属性无法提取控件详细特征的问题, 提高了测试效率.

ARP 机制提出后, Android 平台提供了权限检查 API, 以供应用在运行过程中实时检查权限状态. 应用在调用受权限保护的 API 前, 应通过调用权限检查 API 以判断其是否具有所需权限. 缺失或者错误使用权限检查 API 将导致应用出现 ARP 错误. 因此, 应用发布前, 应被充分测试, 以发现并修复其中的 ARP 错误. 尽管上述 Android 应用 GUI 测试技术在检测应用错误或实现高覆盖率方面取得了较好效果, 但它们在设计时通常未有针对性地考虑 ARP 机制, 无法对应用内的权限相关行为进行有效探索, 因此无法有效检测 ARP 错误. 目前, 已有一些工作关注到应用中的 ARP 错误, 并提出了不同检测技术, 根据是否运行被测应用, 主要可分为静态和动态检测两类.

5.2 ARP 错误静态检测方法

基于静态分析的 ARP 错误检测方法主要使用数据流分析等方式来检测应用中潜在的 ARP 错误.

revDroid[8]用于分析权限撤销给应用带来的不良影响 (尤其是应用崩溃), 其利用 FlowDroid 和 Soot 分析获取被测应用的函数调用图, 并对函数调用图进行可达性分析, 以帮助开发人员在开发期间发现未处理的权限撤销. Huang 等人[38]的工作旨在检测固执的权限请求行为, 即如果用户不授予应用所需的危险权限, 应用将简单地退出, 拒绝提供任何功能该方法首先对固执的权限请求特征进行建模, 随后对调用图进行可达性分析, 以分析应用的权限请求行为. RTPDroid[39]用于检测由 Android 运行时权限机制所带来的恶意行为和错误. 首先, 定位应用中所有敏

感操作 (如调用受权限保护 API); 然后, 基于上下文为敏感操作构建调用图; 最后, 在调用图的范围内, 执行过程间分析, 搜索敏感操作相应的检查和请求语句, 以确定是否出现错误. Wang 等人[4]提出了基于静态分析的权限误用分析工具 APER, 用于检测应用内 ARP 错误. 首先, 遍历应用的调用图, 从危险 API 和权限检查 API 的调用站点执行反向分析, 以提取这些 API 的所有可能调用上下文; 然后, 对应用的程序间控制流图进行分析; 最后, 利用其他组件生成的 API 调用上下文和数据流来定位错误. Gamba 等人[40]进行了大规模的 Android 应用权限使用情况实证研究, 发现当前权限使用中存在的应用自定义权限不透明等相关问题. 为此, 作者设计并实现了两个静态分析工具 permissionTracer 和 permissionTainter, 分别用于检测 Android 应用中潜在的自定义权限和由于使用自定义权限而导致的隐私泄露问题. Li 等人[41]发现此前工作仅关注从 Android API 框架中提取 API 保护映射 (受权限保护的 API 及其对应的权限), 未考虑 Android 本地库 (native library) 中的 API 保护映射, 使用这种不完整的 API 保护映射检测 Android 应用中的安全漏洞时会导致误报率较高. 为此, 他们对 Android 本地库中的 API 保护映射进行了分析, 以提供更完整和准确的规范, 并据此开发了跨语言静态分析工具 NatiDroid, 提高了检测 Android 应用中与权限相关漏洞的能力.

由于无需运行被测应用, 静态检测方法能快速发现应用内潜在的疑似 ARP 错误, 但是其检测结果可能存在大量误报, 且仍需通过开发人员手动验证, 开销较大. RPBDroid 将静态分析结果中可能会导致 ARP 错误的 API 作为目标, 制导自动化测试, 帮助开发人员快速确认 ARP 错误, 并生成测试脚本. ARP 错误静态检测方法的准确率会影响 RPBDroid 性能.

5.3 ARP 错误动态检测方法

除上述静态检测方法外, 已有一些自动化测试工具考虑到了 ARP 机制, 并在测试过程中引入了对应用权限的操作, 可被用于检测应用内 ARP 错误.

部分相关工作旨在通过动态探索被测应用, 以发现应用内潜在的 ARP 错误. 例如, SetDroid[10]用于检测 Android 应用中与系统设置相关的错误, 例如: 网络、GPS、权限、语种、时间格式等. SetDroid 会在两台相同的设备上并行运行相同的被测应用. 在测试过程中, SetDroid 在一台设备上动态生成一组事件序列, 并同时在另一台设备上执行已插入系统设置更改的同一事件序列. 最后, 通过对比两台设备上的 GUI 布局来判断被测应用中是否存在与系统设置相关的错误. SetDroid 支持的设置更改事件包含了权限的授予或撤销, 因此它可用于检测 Android 应用内 ARP 错误. 与本文工作不同的是, SetDroid 会在事件序列的随机位置插入权限更改操作, 而 RPBDroid 会在操作目标控件前进行权限更改操作. 此外, SetDroid 会对应用所需的所有权限进行操作, 而 RPBDroid 仅对可能导致错误的权限进行操作.

另一类工作关注帮助开发人员在各种权限组合下充分测试 Android 应用, 保证应用的健壮性. 例如, PATDroid[42]对被测应用及其测试套件执行混合程序分析, 确定在哪些权限组合上执行哪些测试用例, 以减少测试资源的消耗. PATDroid 减少了测试工作量, 实现了与在所有权限组合下彻底测试应用相当的代码覆盖率和错误检测能力. 但是, 它需要被测应用的测试用例作为输入, 且其性能严重依赖于测试用例质量, 如果无法提供测试用例或测试用例与权限无关, PATDroid 的帮助有限. 与 PATDroid 相似, PermDroid[9]首先为应用构建静态状态转换图, 然后利用静态状态转换图制导, 以在相关权限组合下测试权限相关行为. 与 RPBDroid 不同, 上述工作旨在充分测试 Android 应用内的权限相关行为, 以实现权限相关行为的高代码覆盖率, 而 RPBDroid 是利用静态检测结果以快速检测出 ARP 错误. 此外, PATDroid 需要输入被测应用测试用例, PermDroid 需要静态构建被测应用状态转换图, 而 RPBDroid 可自动生成测试用例, 无需人工设计测试用例, 且通过动态方式构建更加精准的状态转换图.

6 结论与展望

本文提出了一种快速检测 Android 应用中 ARP 错误的方法. 该方法动态构建被测应用的 STG, 并结合静态分析结果更有针对性地测试被测应用, 以快速检测出应用中 ARP 错误. 实验结果表明, 基于该方法实现的原型工具 RPBDroid 检测 ARP 错误的效率高于 ARP 错误动态检测工具 SetDroid、PermDroid 和通用 Android 应用测试工

具 APE. 目前, RPBDroid 中仅将应用崩溃作为检测 ARP 错误的测试预言, 而权限误用也会导致应用功能异常等其他类型的 ARP 错误. 在未来研究工作中, 我们计划进一步扩展检测 ARP 错误的测试预言, 使 RPBDroid 能够检测到更多类型的 ARP 错误.

References:

[1] Google. Permissions on Android. 2022 (in Chinese). https://developer.android.com/guide/topics/permissions/overview

[2] Andriotis P, Sasse MA, Stringhini G. Permissions snapshots: Assessing users' adaptation to the Android runtime permission model. In: Proc. of the 2016 IEEE Int'l Workshop on Information Forensics and Security. Abu Dhabi: IEEE, 2016. 1–6. [doi: 10.1109/WIFS.2016.7823922]

[3] Miao XC, Wang R, Xu L, Zhang WF, Xu BW. Security analysis for Android applications using sensitive path identification. Ruan Jian Xue Bao/Journal of Software, 2017, 28(9): 2248–2263 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5177.htm [doi: 10.13328/j.cnki.jos.005177]

[4] Wang SN, Wang YB, Zhan X, Wang Y, Liu YP, Luo XP, Cheung SC. APER: Evolution-aware runtime permission misuse detection for Android APPs. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 125–137. [doi: 10.1145/3510003.3510074]

[5] Google. APP permissions best practices. 2022 (in Chinese). https://developer.android.com/training/permissions/usage-notes

[6] Wang Y, Wang YB, Wang SN, Liu YP, Xu C, Cheung SC, Yu H, Zhu ZL. Runtime permission issues in Android APPs: Taxonomy, practices, and ways forward. IEEE Trans. on Software Engineering, 2023, 49(1): 185–210. [doi: 10.1109/TSE.2022.3148258]

[7] Dilhara M, Cai HP, Jenkins J. Automated detection and repair of incompatible uses of runtime permissions in Android APPs. In: Proc. of the 5th Int'l Conf. on Mobile Software Engineering and Systems. Gothenburg: ACM, 2018. 67–71. [doi: 10.1145/3197231.3197255]

[8] Fang ZR, Han WL, Li D, Guo ZQ, Guo DH, Wang XS, Qian ZY, Chen H. revDroid: Code analysis of the side effects after dynamic permission revocation of Android APPs. In: Proc. of the 11th ACM on Asia Conf. on Computer and Communications Security. Xi'an: ACM, 2016. 747–758. [doi: 10.1145/2897845.2897914]

[9] Yang SH, Zeng ZG, Song W. PermDroid: Automatically testing permission-related behaviour of Android applications. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 593–604. [doi: 10.1145/3533767.3534221]

[10] Sun JL, Su T, Li JX, Dong Z, Pu GG, Xie T, Su ZD. Understanding and finding system setting-related defects in Android APPs. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 204–215. [doi: 10.1145/3460319.3464806]

[11] Gu TX, Sun CN, Ma XX, Cao C, Xu C, Yao Y, Zhang QR, Lu J, Su ZD. Practical GUI testing of Android applications via model abstraction and refinement. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 269–280. [doi: 10.1109/ICSE.2019.00042]

[12] Mahmud T, Che MR, Yang GW. Android compatibility issue detection using API differences. In: Proc. of the 2021 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2021. 480–490. [doi: 10.1109/SANER50967.2021.00051]

[13] Li JN, Hu K, Li T, Tang L. A FSM-based hardware monitoring technology for packet processing. Ruan Jian Xue Bao/Journal of Software, 2016, 27: 50–57 (in Chinese with English abstract). http://www.jos.org.cn/jos/article/abstract/16018?st=search

[14] Liu P, Miao HK, Zeng HW, Mei J. DFSM-based minimum test cost transition coverage criterion. Ruan Jian Xue Bao/Journal of Software, 2011, 22(7): 1457–1474 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3872.htm [doi: 10.3724/SP.J.1001.2011.03872]

[15] Lei B, Wang LZ, Bu L, Li XD. Robustness testing for components based on state machine model. Ruan Jian Xue Bao/Journal of Software, 2010, 21(5): 930–941 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3544.htm [doi: 10.3724/SP.J.1001.2010.03544]

[16] Li YC, Yang ZY, Guo Y, Chen XQ. DroidBot: A lightweight UI-guided test input generator for Android. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering Companion. Buenos Aires: IEEE, 2017. 23–26. [doi: 10.1109/ICSE-C.2017.8]

[17] Liu Z, Chen CY, Wang JJ, Huang YK, Hu J, Wang Q. Nighthawk: Fully automated localizing UI display issues via visual understanding. IEEE Trans. on Software Engineering, 2022, 49(1): 403–418. [doi: 10.1109/TSE.2022.3150876]

[18] Jin HJ, Liu MY, Dodhia K, Li YC, Srivastava G, Fredrikson M, Agarwal Y, Hong JI. Why are they collecting my data? Inferring the purposes of network traffic in mobile APPs. Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2018, 2(4): 173. [doi: 10.1145/3287051]

[19] Liu Z, Chen CY, Wang JJ, Huang YK, Hu J, Wang Q. Owl eyes: Spotting UI display issues via visual understanding. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 398–409. [doi: 10.1145/3324884.3416547]

[20] Shuba A, Markopoulou A. NoMoATS: Towards automatic detection of mobile tracking. Proc. on Privacy Enhancing Technologies, 2020, 2020(2): 45–66. [doi: 10.2478/popets-2020-0017]

[21] Zhao Y, Yu TT, Su T, Liu Y, Zheng W, Zhang JZ, Halfond WGJ. ReCDroid: Automatically reproducing Android application crashes from bug reports. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 128–139. [doi: 10.1109/ICSE.2019.00030]

[22] Google. UI/Application Exerciser Monkey. 2022. https://developer.android.com/studio/test/monkey

[23] Mao K, Harman M, Jia Y. Sapienz: Multi-objective automated testing for Android applications. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. Saarbrücken: ACM, 2016. 94–105. [doi: 10.1145/2931037.2931054]

[24] Cheng HL, Tang EY, Yu CZ, Zhang CC, Chen X, Wang LZ, Bu L, Li XD. Approach of sketch-guided GUI testing for mobile APP. Ruan Jian Xue Bao/Journal of Software, 2020, 31(12): 3671–3684 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5873.htm [doi: 10.13328/j.cnki.jos.005873]

[25] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android APPs. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg: ACM, 2013. 224–234. [doi: 10.1145/2491411.2491450]

[26] Song W, Qian XX, Huang J. EHBDroid: Beyond GUI testing for Android applications. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana: IEEE, 2017. 27–37. [doi: 10.1109/ASE.2017.8115615]

[27] Su T, Meng GZ, Chen YT, Wu K, Yang WM, Yao Y, Pu GG, Liu Y, Su ZD. Guided, stochastic model-based GUI testing of Android APPs. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 245–256. [doi: 10.1145/3106237.3106298]

[28] Amalfitano D, Fasolino AR, Tramontana P, De Carmine S, Memon AM. Using GUI ripping for automated testing of Android applications. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. Essen: IEEE, 2012. 258–261. [doi: 10.1145/2351676.2351717]

[29] Lai DL, Rubin J. Goal-driven exploration for Android applications. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 115–127. [doi: 10.1109/ASE.2019.00021]

[30] Ls Z, Chen CY, Wang JJ, Su YH, Wang Q. NaviDroid: A tool for guiding manual Android testing via hint moves. In: Proc. of the 44th ACM/IEEE Int'l Conf. on Software Engineering: Companion Proc. Pittsburgh: IEEE, 2022. 154–158. [doi: 10.1145/3510454.3516848]

[31] Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of Android APPs. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. Indianapolis: ACM, 2013. 641–660. [doi: 10.1145/2509136.2509549]

[32] Lv ZW, Peng C, Zhang Z, Su T, Liu K. Fastbot2: Reusable automated model-based GUI testing for Android enhanced by reinforcement learning. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 135. [doi: 10.1145/3551349.3559505]

[33] Koroglu Y, Sen A, Muslu O, Mete Y, Ulker C, Tanriverdi T, Donmez Y. QBE: QLearning-based exploration of Android applications. In: Proc. of the 11th IEEE Int'l Conf. on Software Testing, Verification and Validation. Västerås: IEEE, 2018. 105–115. [doi: 10.1109/ICST.2018.00020]

[34] Pan MX, Huang A, Wang GX, Zhang T, Li XD. Reinforcement learning based curiosity-driven testing of Android applications. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 153–164. [doi: 10.1145/3395363.3397354]

[35] Vuong TAT, Takada S. A reinforcement learning based approach to automated testing of Android applications. In: Proc. of the 9th ACM SIGSOFT Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. Lake Buena Vista: ACM, 2018. 31–37. [doi: 10.1145/3278186.3278191]

[36] Romdhana A, Merlo A, Ceccato M, Tonella P. Deep reinforcement learning for black-box testing of Android APPs. ACM Trans. on Software Engineering and Methodology, 2022, 31(4): 65. [doi: 10.1145/3502868]

[37] Cai LZ, Wang J, Chen MG, Wang JL. Reinforcement learning application testing method based on multi-attribute fusion. In: Proc. of the 9th Int'l Conf. on Dependable Systems and Their Applications. Wulumuqi: IEEE, 2022. 24–33. [doi: 10.1109/DSA56465.2022.00013]

[38] Huang JM, Huang WC, Miao FY, Xiong Y. Detecting stubborn permission requests in Android applications. In: Proc. of the 4th Int'l Conf. on Big Data Computing and Communications. Chicago: IEEE, 2018. 84–89. [doi: 10.1109/BIGCOM.2018.00020]

[39] Zhang J, Tian C, Duan ZH, Zhao L. RTPDroid: Detecting implicitly malicious behaviors under runtime permission model. IEEE Trans. on Reliability, 2021, 70(3): 1295–1308. [doi: 10.1109/TR.2021.3078628]

[40] Gamba J, Feal Á, Blazquez E, Bandara V, Razaghpanah A, Tapiador J, Vallina-Rodriguez N. Mules and permission laundering in Android: Dissecting custom permissions in the wild. IEEE Trans. on Dependable and Secure Computing, 2023: 1–18. [doi: 10.1109/TDSC.2023.3288981]

[41] Li CR, Chen X, Sun RX, Xue MH, Wen S, Ahmed ME, Camtepe S, Xiang Y. Cross-language Android permission specification. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 772–783. [doi: 10.1145/3540250.3549142]

[42] Sadeghi A, Jabbarvand R, Malek S. PATDroid: Permission-aware GUI testing of Android. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 220–232. [doi: 10.1145/3106237.3106250]

附中文参考文献:

[1] Google. Android 中的权限. 2022. https://developer.android.com/guide/topics/permissions/overview

[3] 缪小川, 汪睿, 许蕾, 张卫丰, 徐宝文. 使用敏感路径识别方法分析安卓应用安全性. 软件学报, 2017, 28(9): 2248–2263. http://www.jos.org.cn/1000-9825/5177.htm [doi: 10.13328/j.cnki.jos.005177]

[5] Google. 应用权限最佳做法. 2022. https://developer.android.com/training/permissions/usage-notes

[13] 厉俊男, 胡锴, 李韬, 唐路. 一种基于状态机的硬件分组处理监测技术. 软件学报, 2016, 27: 50–57. http://www.jos.org.cn/jos/article/abstract/16018?st=search

[14] 刘攀, 缪淮扣, 曾红卫, 梅佳. 确定性有限状态机的最小测试成本迁移覆盖准则. 软件学报, 2011, 22(7): 1457–1474. http://www.jos.org.cn/1000-9825/3872.htm [doi: 10.3724/SP.J.1001.2011.03872]

[15] 雷斌, 王林章, 卜磊, 李宣东. 基于状态机模型的构件健壮性测试. 软件学报, 2010, 21(5): 930–941. http://www.jos.org.cn/1000-9825/3544.htm [doi: 10.3724/SP.J.1001.2010.03544]

[24] 成浩亮, 汤恩义, 玉淳舟, 张初成, 陈鑫, 王林章, 卜磊, 李宣东. 一种手绘制导的移动应用界面测试方法. 软件学报, 2020, 31(12): 3671–3684. http://www.jos.org.cn/1000-9825/5873.htm [doi: 10.13328/j.cnki.jos.005873]

林高毅(1998－), 男, 硕士生, 主要研究领域为智能软件分析及测试技术.

陈翔(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件缺陷预测, 软件缺陷定位, 组合测试.

崔展齐(1984－), 男, 博士, 教授, CCF 高级会员, 主要研究领域为智能化软件工程, 可信人工智能.

郑丽伟(1979－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为需求软件工程, 社交网络, 数据质量增强.