引用格式:李阳,文廷科,马慧敏,等. 针对 Android 应用组件间通信的模糊测试技术研究[J]. 信息对抗技术,2024, 3(4):81-94. [LI Yang, WEN Tingke, MA Huimin, et al. Research on fuzzing of inter-component communication in Android applications[J]. Information Countermeasure Technology, 2024, 3(4):81-94. (in Chinese)]

针对 Android 应用组件间通信的模糊测试技术研究

李 阳[1,2],文廷科[1,2],马慧敏[1,2],王瑞鹏[1,2],李倩玉[1,2],潘祖烈[1,2*]

(1. 国防科技大学电子对抗学院,安徽合肥 230037; 2. 网络空间安全态势感知与评估安徽省重点实验室,安徽合肥 230037)

摘 要 Intent 是 Android 应用中最常用的组件间相互通信的载体。然而,如果应用组件对 Intent 处理不当,极有可能导致应用异常甚至崩溃。以 Android 应用的各个组件为研究对象,提出了一种通过构造 Intent 对象来测试 Android 应用组件间通信健壮性的模糊测试方法。首先分析应用组件的注册文件,提取需要测试的组件及其相关信息。然后,反编译 APK 源文件,获得目标组件的源代码并提取 Intent 的附加信息。接着,基于状态压缩批量生成目标组件的 Intent 测试用例用于自动化测试,并监控目标组件的运行日志来获取其运行状态反馈,据此判断应用组件在响应 Intent 时是否发生异常。最后,基于相似度匹配的错误日志去重算法,准确地将同一缺陷生成的错误日志归为一类,降低人工分析的工作量。实验表明,所提方法相较于现有前沿研究 Hwacha,能够少生成 9%的测试用例,多发现 14%的程序异常,并通过去重算法显著降低了需要人工研判错误类别的工作量。

关键词 模糊测试;软件测试;Android 安全;自动化测试

中图分类号 TP 393 **文章编号** 2097-163X(2024)04-0081-14

文献标志码 A **DOI** 10.12399/j.issn.2097-163x.2024.04.006

Research on fuzzing of inter-component communication in Android applications

LI Yang[1,2], WEN Tingke[1,2], MA Huimin[1,2],
WANG Ruipeng[1,2], LI Qianyu[1,2], PAN Zulie[1,2*]

(1. College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China;
2. Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China)

Abstract Intent is the most commonly used carrier for inter-component communication in Android applications. However, if application components handle Intent improperly, it is very likely to cause abnormalities or even crashes. Taking the various components of Android applications as the research object, a fuzzing method was proposed to test the robustness of inter-component communication by constructing Intent objects. Firstly, the registration files of application components were analyzed to extract the components and their related information that need to be tested. Then, the APK source file was decompiled to obtain the source code of the target component and extract additional information of Intent. Next, based on state compression, a batch of Intent test cases for the target component were generated for

收稿日期:2023-09-07 **修回日期**:2024-03-01

通信作者:潘祖烈,E-mail:panzulie17@nudt.edu.cn

基金项目:国家重点研发计划项目(2022YFB3102900)

automated testing, and the running logs of the target component were monitored to obtain feedback on its operational status, thereby determining whether an exception occurs when the application component responds to Intent. Finally, based on an error log deduplication algorithm with similarity matching, errors generated by the same defect were accurately classified into one category, reducing the workload of manual analysis. Experiments show that compared with the existing cutting-edge research Hwacha, the proposed method can generate 9% fewer test cases and discover 14% more program exceptions, and significantly reduce the workload of manually determining error categories through the deduplication algorithm.

Keywords fuzzing; software testing; Android security; automated testing

0 引言

作为移动互联网时代的流量入口,移动应用的数量越来越多,更新迭代的速度越来越快,大量的开发人员涌入移动开发行业。由于开发人员自身的水平参差不齐,以及项目进度的压力,导致许多应用中不可避免地存在着各种潜在的缺陷甚至漏洞。根据友盟的研究报告[1],2020 年度 Android 应用整体的崩溃率为 0.32%,且数量上居多的非头部应用的崩溃率高于平均水平。从日均活跃用户数量(daily active user, DAU)的角度来看,DAU 低于 100 万的应用崩溃率超过 0.78%,而 DAU 低于 5 万的应用崩溃率超过 1.15%,远远超过平均水平。随着大众对移动应用的依赖程度越来越高,应用本身因为崩溃等原因导致不可用的情况不仅会严重影响用户的使用体验,而且还有可能导致重要信息的丢失或泄露,进而造成其他方面的损失。因此,如何对应用进行有效的测试,及时发现应用的缺陷和漏洞,避免造成更大的损失,是一个重要且热门的研究方向。

Intent 是 Android 中最常用的组件间相互通信的信息载体,可以用于启动或者调用 Android 4 大组件中的 Activity、Service 和 Broadcast Receiver。Intent 不仅包含指导系统完成调用操作的相关信息,还可以通过 Extra 附加字段在组件间传递数据。使用 Intent 机制可以将 Android 应用的各个组件耦合起来,开发人员也可以通过设置 exported 属性将组件对外暴露,使得组件可以接收并响应来自自身应用之外的 Intent,与其他应用以及 Android 系统进行交互,提高组件功能的重用性。当一个组件对外暴露时,它可能会接收到大量非预期的 Intent 对象。这些非预期 Intent 对象可能来自没有严格遵循开发规范的第三方程序,也有可能来自恶意应用。如果组件的开发人员没有设计好响应外来 Intent 的逻辑,使得组件无法对非预期组件进行合理的处理,那么可能会导致应用运行异常,甚至崩溃。

1 相关研究

1.1 基于 Android 事件的测试技术研究

点击或者滑动屏幕是用户与 Android 应用交互的主要形式,以屏幕事件为主的各种用户事件驱动着 Android 应用的运行。因此,通过模拟生成用户事件来测试应用的稳健性是一种简单但有效的思路。目前,工业界已经出现了一些成熟的基于用户事件的自动化测试工具。

Monkey[2]是 Google 开发的一款自动化测试工具,可以在模拟器或者真实的物理设备上运行。Monkey 可以通过生成伪随机用户事件流(例如点击、轻触或手势)以及许多其他系统级事件的方式,来对正在开发的应用进行压力测试。但是,也正因为它的生成策略是随机的,所以它的测试并没有针对性,效率较低。

为了使得开发人员可以通过自定义的测试用例来对应用进行测试,Google 又推出了 MonkeyRunner[3]。MonkeyRunner 支持通过编写 Python 代码来操纵 Android 应用及其安装包,同时通过应用程序编程接口(application programming interface, API)发送特定命令和事件来控制设备或者模拟器。MonkeyRunner 提供了多设备控制、功能测试和回归测试的功能,能够以屏幕截图的形式记录测试结果,方便测试人员进行比较。

UI Automator[4]是 Google 开发的另外一款 UI 测试框架,适用于整个系统上以及多个已安装

应用间的跨应用功能界面测试。UI Automator 测试框架提供了一组 API，用于构建在用户应用和系统应用上执行交互的界面测试。该框架非常适合编写黑盒式自动化测试，此类测试的测试代码不依赖于目标应用的内部实现细节。

现有的基于 Android 事件的测试工具多聚焦于 UI 测试，这是因为 UI 组件既是应用运行结果反馈的主要形式，也是用户与应用进行交互的主要媒介。UI 组件的稳定性和功能性能够在一定程度上反映整个应用的健壮性。但是，上述的这些测试工具多通过生成事件流来对应用进行黑盒测试，并没有使用程序分析技术，无法探测应用深层的逻辑缺陷。

1.2　基于 Intent 的 Android Fuzz 技术研究

Intent 是 Android 中最为常用的用于组件间相互通信的信息载体，通过合理设置组件的属性，可以使组件接收并响应来自自身应用之外的 Intent 对象。基于 Intent 对 Android 应用进行模糊测试，可以探测组件在响应外来 Intent 过程中可能存在的潜在缺陷。

JarJarBinks[5]能够自动化提取出目标应用的 Intent-Filter，并基于这些信息和一些特定的策略，生成合法与半合法的 Intent 对象，以实现对应用的测试。但是，JarJarBinks 并没有实现完全的自动化测试，仍然需要使用者的干预。

DroidFuzzer[6]能够自动化识别那些能够接收外部多用途互联网邮件扩展类型（multipurpose Internet mail extensions，MIME）数据的 Activity，通过对 MIME 数据的重要字段进行变异生成测试 Intent 对象，来测试这些 Activity 在解析外来 MIME 数据时的稳健性，并发现隐含的漏洞。同时，DroidFuzzer 还包含了动态监测模块，通过覆写 ActivityController 类、解析 tombstone 文件，在 Java 层和 Native 层对 crash 以及其他异常事件进行动态监测。DroidFuzzer 重点关注接收外部 MIME 数据的 Activity，因此它主要测试视频音乐类应用。

IntentFuzzer[7]以 Service 和 Broadcast Receiver 为主要研究对象，能够发现 Android 应用中权限泄露的问题。该工具向目标组件发送精心构造的 Intent 对象，并修改 Android 系统源码来监测当前应用是否因为此 Intent 对象的输入而尝试向 Android 系统申请未被授予的权限，以此作为权限泄露的标志。在构造 Intent 时，IntentFuzzer 会从 DEX 文件的常量池中抽取特定模式的字符串，与 Android 预定义的值一起作为 Action 字段的候选值。对于 Extra 字段，IntentFuzzer 同样修改了 Android 系统源代码，监测应用的 API 调用情况来获取组件尝试解析的 Extra 原语键值对。该方法对 Android 系统源码进行了一定的改动，在测试的过程中获取 Extra 信息，相较于使用程序分析技术，会导致整个测试周期更长，而且可能出现遗漏的情况。

同样旨在发现权限泄露问题，ParaIntentFuzz[8]是一套基于动态任务分配、部署在云计算平台上的 Android 应用并行化测试框架。相较于之前 Android 模糊测试多为单机测试的情况，ParaIntentFuzz 极大地提升了测试效率。此外，文献[8]还提出了一种轻量级的 Extra 信息提取方法，并通过实验验证了 Extra 信息的有效性。但是，文章并没有详细讨论如何深入利用 Extra 信息来改进 Intent 生成策略。

Hwacha[9]创新性地提出了 Intent 范式（Intent specification）这一概念，用于描述被测试组件所能响应的 Intent 在内部结构上的特性，并以此作为 Intent 测试用例的生成模板。同时，Intent 范式也在逻辑上将模糊测试工具的程序分析模块和用例生成模块解耦，遵循这一范式的分析器和生成器可以相互组合，便于探索和实现更为高效的模糊测试工具。但是，Hwacha 的程序分析和测试用例生成算法相对简单，使得构建的 Intent 范式包含的信息较少、生成的测试用例质量较低。因此，Hwacha 在实际测试中的效率较低。

1.3　现有研究存在的不足

通过分析相关文献和工具，可以总结得到现有研究和工具存在以下不足之处：

1）Android 应用分析不充分。AndroidManifest.xml 文件中包含了 Android 应用组件的注册信息，以及组件的部分属性定义，例如 Action 和 Category 属性。但是，组件响应的 Intent 的 Extra 字段数据并不包含在其中，只能通过分析源代码得出。现有研究大多仅尝试提取了 APK 文件中的 AndroidManifest.xml 文件，而并没有尝试使用已有的 Android 应用分析技术来分析源代码。通过这种方式提取出来的信息是不完善的，将会直接影响到后续的建模效果和测试用例

的质量。

2）Intent 建模与测试用例生成方法不完善。现有研究大多没有尝试对关键研究对象进行建模的工作，从而导致这些研究工作的应用分析模块和测试用例生成模块绑定紧密，不方便修改或扩展。Hwacha 完成了对被测试组件所能响应的 Intent 的建模工作，但受限于 Hwacha 简单的程序分析模块，Intent 范式中并没有包含例如 Extra 这类有价值的信息，这导致其建模工作是不完善的。此外，不完善的 Intent 模型还影响了测试用例生成策略的有效性。

3）错误日志去重算法存在缺陷。现有研究大多采用人工分析错误日志的方法来判定问题根源并将缺陷分类，但是当数据量较大时会造成分析工作量大大增加，降低分析效率。Hwacha 实现了一种基于最长公共子序列的算法来计算任意 2 组错误日志之间的相似度，以此为依据将错误日志去重并归类，加快了日志分析的效率。但是，该方法涵盖了较大比例的不必要信息，且忽略了目标组件等直观准确的分类依据，仍然存在一定的优化空间。

针对上述问题，本文从以下 2 个方面开展了研究，并设计实现了针对 Intent 对象的 Android 组件间通信模糊测试原型系统。

1）Intent 建模与基于状态压缩的测试用例生成算法。Android 应用中的所有组件都需要进行注册才能生效，本文提取并分析应用的 AndroidManifest. xml 文件来确定组件中 Intent 对象的基本信息。同时，借助现有成熟的 APK 反编译技术获取组件的 Java 代码并分析提取 Extra 字段数据，用于构建更为完善的 Intent 测试对象[10]。然后，针对应用中的每个组件，建立相对应的 Intent 范式，包含所有可能影响 Intent 响应过程的数据。进一步地，以组件信息和 Intent 范式为基础，本文基于状态压缩设计了高效的测试用例生成算法，用于批量生成高质量的 Intent 测试用例。

2）基于相似度匹配的错误日志去重算法。对同一个测试目标生成大量的测试用例，然后记录并分析日志数据，这是模糊测试的特点之一。因此，同一个缺陷可能在测试过程中产生多条错误日志，如果对其进行逐一分析则会浪费较大的时间和精力[11]。本文设计了一种相似度匹配的去重算法，并结合一定的优化策略，能够准确地将因同一缺陷而产生的错误日志归为一类，从而提高软件缺陷分析工作的效率。

2 方法设计

本文设计并实现了面向 Android 平台应用组件间通信的模糊测试原型系统——iFuzzer。该系统能够分析目标应用的 APK 文件，对应用组件所能响应的 Intent 对象进行建模，然后在此基础上生成测试用例并进行自动化的模糊测试，监控目标应用的运行情况，以检测当前 APK 软件是否存在缺陷问题。最后，本系统还能够分析测试过程中产生的日志信息并去除冗余信息，降低后续人工分析缺陷原因的工作量。iFuzzer 的结构如图 1 所示，主要包含应用分析模块、测试用例生成模块、测试模块和日志分析模块。

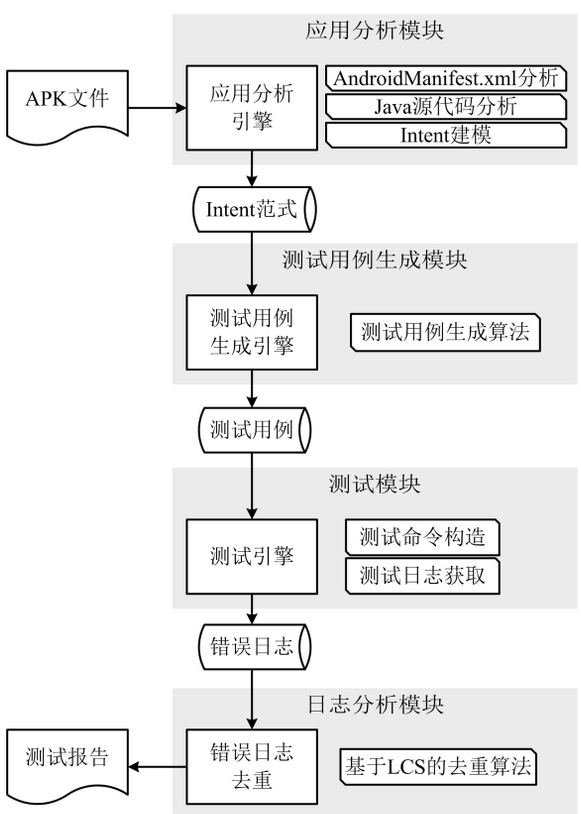

图 1 iFuzzer 结构示意图

Fig. 1 Schematic diagram of iFuzzer's structure

2.1 应用分析模块

应用分析模块主要承担 APK 文件解析和 Intent 建模 2 个任务。该模块接收输入的 Android 应用 APK 文件，使用 jadx[12] 工具解析改文件并得到 AndroidManifest. xml 文件和反编译的 Java 源代码，然后从中提取应用组件的信息并构

建 Intent 范式，为后续的测试用例生成提供支撑。

2.1.1　AndroidManifest.xml 分析

AndroidManifest.xml 文件向 Android 系统描述了所属 Android 应用的基本信息，其中包含了静态注册组件的相关信息。应用分析模块的首要工作就是通过解析 AndroidManifest.xml 文件来确定需要测试的应用组件。

Android 应用包含 Activity、Service、Broadcast Receiver 和 Content Provider 4 大组件。其中，Content Provider 组件本身不通过 Intent 与外部交互[13]，不在测试范围内，因此主要对其余 3 类组件展开研究和测试。之后，需要确定组件是否能够响应外来 Intent。iFuzzer 通过自动分析组件 exported 属性和 Intent-filter 标签实现对目标组件的自动提取。在确定待测目标后，需要对目标组件的 Intent 信息进行进一步分析，抽取其中的 Data 参数和 Extra 字段，定位到组件对应源码，为后续的源码分析提供基础。Data 参数用于声明待操作的数据引用，往往涉及一个真实存在于应用或设备中的资源对象。例如，视频、音频、图片之类的多媒体文件。本文未涉及多媒体文件的构造与测试，因此，不对 Data 参数进行测试。

2.1.2　Java 源代码分析

使用现有成熟的 Android 应用反编译工具 jadx，可以获取到应用的源代码文件，在源码中最值得关注的是 Extra 数据，这也是 iFuzzer 分析的重点数据。Extra 数据在形式上是一系列包含在 Intent 对象中的键值对数据，Android 提供了一系列的 API 供开发者从 Intent 中提取对应的 Extra 数据，这些 API 的名称和对应的 Extra 数据类型的映射关系见表 1 所列。其中，部分 API 在使用时会要求声明默认值，当使用该 API 提取数据失败时会将该默认值作为返回值。因此，iFuzzer 会扫描 Java 源代码提取使用过的 API，确定组件从外来 Intent 中提取到的 Extra 数据，记录这些 Extra 数据的键名、数据类型和默认值。

至此，应用的分析工作已经完成，iFuzzer 从 APK 文件中提取了被测试的组件信息，包括组件名称 Component Name、动作类型 Action、组件类型 Category 和附加数据 Extra 的信息。

事实上，Intent 还包含有 Flags 信息，它设置 Android 系统中 Activity 组件的启动模式，但不影响组件解析 Intent 逻辑，因此不纳入研究范围。后续将基于前 4 类信息进行 Intent 建模工作。

表 1　与 Extra 信息提取相关的 API 映射表

Tab.1　API mapping table related to Extra information extraction

序号	API	参数类型
1	getStringExtra(String name)	String
2	getBooleanExtra(String name, boolean defaultValue)	boolean
3	getIntExtra(String name, int defaultValue)	int
4	getLongExtra(String name, long defaultValue)	long
5	getFloatExtra(String name, float defaultValue)	float
6	getLongArrayExtra(String name)	long[]
7	getFloatArrayExtra(String name)	float[]
8	getIntArrayExtra(String name)	int[]

2.1.3　Intent 建模

Intent 是本文研究的核心线索，每个模块都围绕着它展开。每个组件对应着一个抽象的 Intent，表示该组件所能响应的所有 Intent 的集合。同时，在进行模糊测试之前又需要先生成大量具体的 Intent 对象作为测试用例。抽象的 Intent 串联了测试工具的应用分析模块和测试用例生成模块。现有的多数测试工具并没有对 Intent 建立规范化的描述模型，使得它们的应用分析模块和测试用例生成模块联系紧密，对其中一方的修改都很有可能导致对另一方的调整。同时，难以通过替换模块的方式来拓展测试工具的能力。

借鉴 Hwacha 的设计思路，本文设计了 Intent 范式来对组件所能响应的所有 Intent 集合进行规范化的描述。应用分析模块提取到的组件信息将会被用于构建 Intent 范式，同时针对该组件的测试用例也将基于该组件对应的 Intent 范式来进行生成。进一步地，通过在应用分析模块和测试用例生成模块之间加入 Intent 范式，可以有效地将两者解耦。图 2 为 Intent 范式解耦分析模块和测试用例生成模块示意图，任何遵循 Intent 范式的应用分析模块和测试用例生成模块可以自由地匹配，尝试更多的算法组合，而无须考虑算法之间是否存在相互作用。

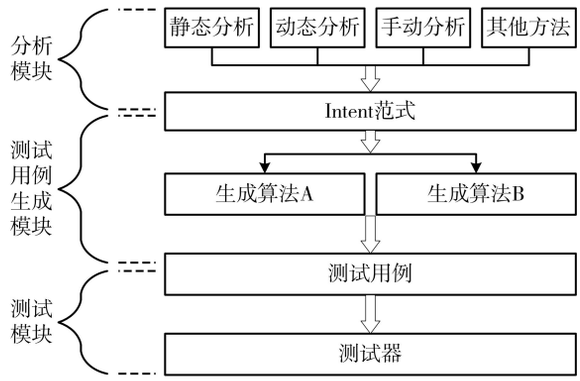

图 2 Intent 范式解耦分析模块和测试用例生成模块示意图

Fig. 2 Schematic diagram of Intent specification decoupling analysis module and test case generation module

Intent 范式的定义如图 3 所示。Intent 范式在形式上由一系列字段组成，分别为组件名称 cmp、动作类型 act、组件类型 cat 和附加数据 exs。cmp 字段指明了组件的名称和类型，act 字段以列表的形式指明了组件响应的所有 Intent 动作类型，cat 字段以列表的形式指明了组件为自身声明的类型。此外，exs 字段以键值对列表的形式指明了 Intent 对象应当携带的数据，包括数据的键名、默认键值和类型。

根据定义，LineSchemeServiceActivity 组件生成的 Intent 范式如图 4 所示。从该范式中，可以直观而清晰地读出以下信息：LineSchemeServiceActivity 组件是一个 Activity，所属包名为 jp.naver.line.android.activity.schemeservice，所属应用的标识符为 jp.naver.line.android；该组件可以响应动作类型为 android.intent.action.VIEW 的 Intent；该组件声明的类型包括 android.intent.category.BROWSABLE 和 android.int ent.category.DEFAULT；该组件尝试从外来 Intent 提取至少 2 个 String 类型的 Extra 数据来使用，对应的键名分别为 invited_groupid 和 notification_center_revision，因为 String 类型数据在提取时不需要声明默认值，这里将键名作为默认值。后续针对 LineSchemeServiceActivity 组件的测试用例将会基于该范式来生成。

```
{
 cmp = TYPE COMPONENTNAME
 act = [ ACTION, ACTION, ... ]
 cat = [ CATEGORY, CATEGORY, ... ]
 exs = [ STR = EXTRAVALUE, STR = EXTRAVALUE, ... ]
}

TYPE = Activity | BroadcastReceiver | Service
COMPONENTNAME = STR / (STR | .STR)
ACTION = STR
CATEGORY = STR
EXTRAVALUE = String STRING | boolean BOOL | int INT | long LONG | float FLOAT | long[]
     LONG, ..., LONG | float[] FLOAT, ..., FLOAT | int[] INT, ..., INT

STR = (a-zA-Z0-9_.)*
```

图 3 Intent 范式定义

Fig. 3 Definition of Intent specification

```
{
 cmp=Activity jp.naver.line.android/jp.naver.line.android.activity.schemeservice.
     LineSchemeServiceActivity
 act=[ android.intent.action.VIEW ]
 cat=[ android.intent.category.BROWSABLE, android.intent.category.DEFAULT ]
 exs=[ invited_groupid = String "invited_groupid", extra_from_push = boolean false,
     notification_center_revision = String "notification_center_revision" ]
}
```

图 4 LineSchemeServiceActivity 组件生成的 Intent 范式

Fig. 4 Intent specification generated by LineSchemeServiceActivity component

2.2 测试用例生成模块

在对每个组件都建立了 Intent 范式之后，需要基于这些范式为组件生成对应的 Intent 测试用例。一个 Intent 对象包含 Component Name、Action、Category、Extra、Data 和 Flags 这 6 类数据。据前文所述，Data 和 Flags 数据不纳入研究范围，测试用例生成算法主要围绕前 4 类数据展开。

2.2.1 变异规则

Intent 范式中的 cmp 明确了目标组件类型和包路径等不可缺少的参数。

Intent 只能设置一个 Action 值，它指明了要执行的通用操作类型。对应地，Intent-Filter 中设置的 Action 指明了该组件能够响应的操作类型。组件所响应的 Intent 的 Action 参数必须是其 Intent-Filter 声明的所有 Action 值中的一种，或者为空。因此，在生成 Intent 测试用例时，需要遍历所有的 Action 候选值，并尝试将 Intent 的 Action 设置为空。

组件的 Intent 和 Intent-Filter 都可以设置多个 Category 值，但是 Intent-Filter 中设置的 Category 值需要完全包含 Intent 设置的 Category 值才能响应该 Intent。因此，在生成 Intent 测试用例时，需要尝试 Intent-Filter 中 Category 值的所有子集。

Extra 本质上是一系列键值对数据，可以用于放置需要在组件间传递的各种类型的数据。相比较于 Action 和 Category，Extra 的信息容量更大，对其进行合理的变化有利于探测更多的代码块。对此，本文提出以下 3 条变异规则：

1）组合优于随机。现有研究在设计生成策略时，会尝试生成一个新的 Extra 数据，其键名、键值、数据类型完全随机。但是这种策略的效率是极低的，因为随机生成的键名恰好是组件所要提取 Extra 数据的键名之一的概率是非常小的，在这个前提下数据类型和键值可以探测新代码块的概率更小。因此，本文将聚焦于已知的 Extra 信息，尝试遍历所有键值对的所有组合形式，每个 Extra 数据包含缺失和存在 2 种形式。这种策略也应用于 Category 值的生成。

2）数据类型变异。根据 Android 开发文档，在从 Intent 中提取 Extra 数据时，使用的 API 函数需要与要提取的数据类型相对应。但是在添加 Extra 数据构造 Intent 时却并没有类似的要求，所有的添加操作使用的都是 putExtra()函数，即 Extra 数据的添加和提取 API 函数并没有一一对应。事实上，Extra 数据结构是基于 hash 表实现的，添加和提取操作都需要使用键名对 hash 表进行访问。若发送组件实际添加的数据类型和接收组件使用的 API 函数不一致，则会导致接收组件提取的数据类型并非预期，进而导致后续处理中的异常或者错误。因此，本文将尝试改变 Extra 中部分数据的类型，检测组件在处理这种非预期输入时是否会发生异常甚至崩溃。

3）优先设置默认返回值。在从 Intent 中提取 Extra 数据时，部分 API 函数会要求提供默认值作为提取错误时函数的返回值。部分代码会检测提取到的值是否为默认值，从而执行不同的代码逻辑。因此，本文使用代码中提供的默认值作为 Extra 数据实际值的候选值，用于提升代码覆盖率。

综上，对于一个组件的 Intent 范式：假设其包含 a 个 Action 值，加上空值，Action 共有 $a+1$ 种候选值；假设其包含 c 个 Category 值，这些值的集合共有 2^c 个子集，则整个 Category 有 2^c 种候选值；假设其 Extra 中包含 e 个 Extra 数据，每个数据有置空、默认值、随机值、改变数据类型并生成的随机值共计 4 种候选值，则整个 Extra 有 4^e 种候选值。综上所述，应当为该 Intent 范式对应的组件生成共计 $(a+1)\cdot 2^c\cdot 4^e$ 个测试用例。

2.2.2　基于状态压缩的测试用例生成算法

为了批量生成高质量的 Intent 测试用例，本文设计并实现了基于状态压缩的 Intent 测试用例生成算法，见算法 1 所示。其中，第 3 行代码枚举了 Action 的所有候选值，即从 AndroidManifest.xml 文件中提取出的 Action 列表，再加上空值；第 4 行代码枚举了 Category 的所有候选值，即从 AndroidManifest.xml 文件中提取出的 Category 值的所有子集；第 6 行代码开始的 for 语句枚举了 Extra 字段的所有可能形式。将针对上述 3 类值的枚举语句嵌套起来，遍历了 Intent 范式所描述的整个枚举空间，得到了在本算法下对应组件所能响应的 Intent 的所有值。

第 6 行代码开始的 for 语句，基于状态压缩实现了迭代式空间枚举。结合 1.3 小节中所提出的对于 Extra 数据的生成策略，对于 Intent 范式中包含 E 个键值对的 Extra 数据，每个键值对都有置空、默认值、随机值、改变数据类型并生成的随机值共计 4 种候选值，因此 Extra 数据共有 4^E 种形式。状态压缩的基本思路是借助一个整数来描述这 4^E 种形式。

记整数为 $0\leqslant s<4^E$，若以四进制数来看待，则 $s_{(4)}$ 共有 E 个数位（高位不足补 0），每个数位上的数字取值范围为[0,3]。将这 E 个数位对应 E 个键值对数据，则数位上的数字则对应了键值对数据的 4 种候选值。对应到伪代码中，第 6 行中取值范围为$[0,4^E)$的变量 state 是 Extra 中键值对取值组合的压缩表示，第 8 行开始的 while 语句会取出 state 在四进制表示下每个数位上的值，以此选定键值对数据的生成方式。

在实际测试中，可能会出现部分组件的 Intent 范式携带的信息较多，特别是 Extra 中数据较多，导致枚举空间较大，进而导致生成的测试用例数目较大、测试时间不可控。因此，算法

引入了控制生成测试用例的变量 N，表示对于每个组件，最多生成 N 个测试用例。若实际生成的测试用例超过 N，则会随机返回 N 个测试用例。

算法 1 测试用例生成算法

输入：IntentSpec：Intent 范式；N：测试用例最大生成数量
输出：IntentCases：Intent 测试用例

```
cmp = IntentSpec.cmp
type = IntentSpec.type                                          //获取被测目标组件的包名和类型
for act; act ∈ {IntentSpec.actList + null}; do                  //枚举 Action 属性值
  for catList;catList ∈ getAllCombination(IntentSpec.catList); do   //枚举 Category 属性值的所有组合形式
    S = POW(4, IntentSpec.extra.size()
    for state = 0; state < S; state + + do                      //基于状态压缩枚举 Extra 键值所有组合形式
      s = state, extraList = List(), ind = 0
      while s > 0 do                                            //对状态压缩变量进行位运算，确定当前 Extra 键值对的取值
        if s%4 == 1 then
          extraList.append(GetDefault(IntentSpec.extra[ind]))   //使用默认值
        else if s%4 == 2 then
          extraList.append(ChangeValue(IntentSpec.extra[ind]))  //使用随机值
        else if s%4 == 3 then
          extraList.append(ChangeType(IntentSpec.extra[ind]))   //改变数据类型
        else
          s = s/4, ind + +                                      //置空
        end if
      end while
      IntentCases.append(Construct(cmp, type, act, catList, extraList))  //基于包名、类型、Action、Category 和
                                                                         Extrace 的取值构造并添加测试用例
    end for
  end for
return RandomSelect(IntentCases,N)                              //随机返回指定数量的测试用例
```

2.3 测试模块

测试模块的主要工作是使用生成的 Intent 用例对目标应用进行自动化的测试，并监控测试过程中目标应用的表现，通过监控并分析系统日志的方式判断目标应用是否出现错误。若出现错误，则需要记录系统日志以及对应的 Intent 测试用例，用于后续的分析和错误定位。

经过测试用例生成模块，虽然已经可以得到构造明确的 Intent 实例，但是并不能直接使用，需要借助安卓调试桥（Android debug bridge，ADB）命令基于测试实例构造一个 Intent 对象并发送给目标应用。对于日志信息的获取，同样可以使用 Logcat 工具来实时获取。

本文测试模块的整体工作流程如图 5 所示，对于生成的每个测试用例，都使用 ADB 命令构造 Intent 对象并发送到测试设备上，然后监听目标应用的日志信息。当目标应用完成 Intent 解析后，测试模块再通过 ADB 命令结束应用进程，记录相关数据后进入下一轮测试。

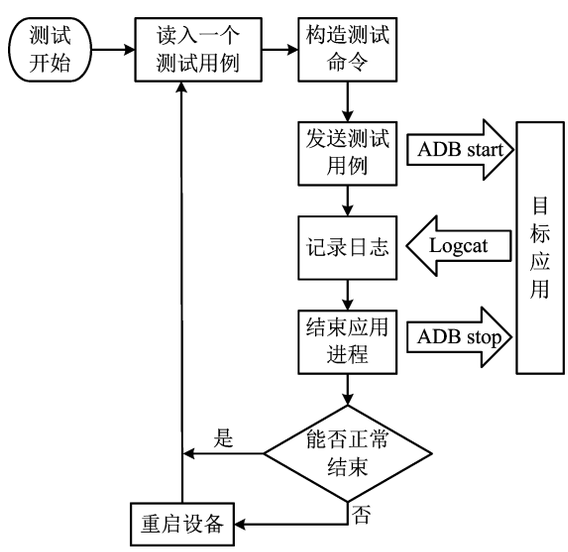

图 5 测试模块工作流程示意图

Fig. 5 Schematic diagram of the test module workflow

在每一轮测试过程中，难免出现非预期的异常情况。这种异常可能是由于当前测试用例触发目标组件的解析逻辑缺陷造成的应用错误而结束，也有可能是应用本身的异常导致应用无法正常运行而结束。当后一种情况发生时，测试模块会尝试重启整个设备来保证测试的正常进行。

2.4　日志分析模块

对于接收到的日志记录，还需要对其进行分析以判断是否存在错误信息，从而判断当前的 Intent 测试用例是否触发了组件内部的缺陷[11]。每次测试获取的一组日志包含若干条日志消息。以表 2 所列的日志为例，该组日志从第 4 条日志消息开始连续抛出了 3 个 Error：第 4 条日志指明发生错误的应用为 com. kakao. story；第 5 条日志指明发生错误的原因为 NullPointerException；第 6 条日志指明发生错误的具体位置是 com. kakao. story. android. widget. Write Article Widget. 组件源代码的第 74 行。可以通过关键字匹配的方法，观察日志信息中是否存在相关字符串来判断组件是否因测试而崩溃。若存在崩溃，则将对应的日志信息记录下来用于后续分析。

但是，通过上述方法得到的仅与错误有关的日志信息，其条目数量仍然可能较大。此外，由于模糊测试会对测试目标进行多次测试，可能使得测试系统就同一个应用缺陷报告了多次错误，从而导致日志信息中存在冗余。因此，还需要对这些日志进行去重处理，将内容相近的日志归为同一类，降低人工分析的工作量。

本文实现了一种基于最长公共子序列（longest common subsequence，LCS）算法的错误日志去重算法，它基于 2 组日志的最长公共子序列长度来计算日志相似度，将相似度超过一定阈值的 2 组日志归为同一类。在后续的人工分析时，分析者可以按照类别一次性完成多组日志的分析工作，而不需要逐个分析所有日志。具体实现过程包含如下步骤：

表 2　Logcat 获取的日志记录片段

Tab. 2　Log snippets obtained by Logcat

序号	优先级	日期和调用时间	进程号	线程号	标记	日志消息
1	Verbose	2023-04-03 14:20:09.599	28423	28423	GCMRegistrar	Registering app com. kakao. story of senders 24220010265
2	Info	2023-04-03 14:20:09.685	686	705	ActivityManager	Start proc 28467: com. android. webview: sandboxed_process0/u0i220 for webview_service com. kakao . story/org. chromium. content. app. SandboxedProcessService0
3	Warning	2023-04-03 14:20:09.750	28423	28494	cr_CrashFileManager	/data/user/0/com. kakao. story/cache/WebView/Crash Reports does not exist or is not a directory
4	Error	2023-04-03 14:20:09.761	28423	28423	AndroidRuntime	Process: com. kakao. story, PID: 28423
5	Error	2023-04-03 14:20:09.761	28423	28423	AndroidRuntime	java. lang. RuntimeException: Unable to start receiver com. kakao. story. android. widget. WriteArticle Widget: java. lang. NullPointerException: Attempt to invoke virtual method'int java. lang. String . hashCode()' on a null object reference
6	Error	2023-04-03 14:20:09.761	28423	28423	AndroidRuntime	at com. kakao. story. android. widget. Write Article Widget. on Receive(SourceFile:74)
7	Info	2023-04-03 14:20:09.891	686	1319	ActivityManager	Process com. kakao. story (pid 28423) has died:cch CEM

1）判断日志错误信息。对于接收的日志记录，根据其是否包含特定字符串来判断是否产生了错误。对于 Activity 组件，需要检查是否出现“ActivityManager：Force finishing activity”字符串；对于 Service 组件，需要检查是否出现“AndroidRuntime：Shutting down VM”字符串；对于 Broadcast Receiver 组件，需要检查是否出现“ActivityManager：Process pid * * * * has died”模式的字符串。

2）日志类别及关键字段选取。对于分析得到的错误日志，可以观察到每组日志还包含多条日志记录，每条日志记录又包含优先级、日期和调用时间、标记、进程号、线程号和日志消息这些字段。然而，并不是日志消息中的所有字段都可以用于计算相似度，也不是一组错误日志中的所有日志消息都有必要参与相似度计算。举例来说，对于日志中的各个字段：日期和调用时间与缺陷无关，只与测试时间相关；进程号和线程号取决于操作系统的分配；优先级只包含一个字符，对相似度计算结果的影响可以忽略。因此，只保留每条日志的消息和标记。对于一组错误日志，每条日志消息都有对应的优先级，因为 Fatal、Error、Warning 级别日志消息与错误事件密切相关，所以对于每组错误日志，只保留这 3 类优先级的日志消息参与相似度计算。

3）相似日志分类。去重算法将每组日志中入选的日志消息和入选的字段值串联起来构成一个字符串，记作日志字符串。然后枚举任意 2 组日志，计算其日志字符串的最长公共子序列长度，并将该长度除以 2 个日志字符串的平均长度作为 2 组日志的相似度。若 2 组日志的相似度超过一定阈值，则将这 2 组日志归为同一类。

基于相似度匹配的日志去重算法的具体流程见算法 2 所示。首先，对于所有需要去重并归类的日志，都将被 convert2str()函数转换为日志字符串。其次，枚举任意 2 个日志字符串，计算它们的最长公共子序列长度，在此基础上计算 2 个日志字符串的相似度，即 2 组日志的相似度。最后，若相似度大于阈值，则认为 2 组日志由同一个缺陷引起，应划分为同一类。

同时，算法还引入目标组件名称来辅助判断。从每组日志对应的 ADB 测试命令中提取目标组件名称，仅当 2 组日志都属于对同一个组件的测试时，才计算相似度来判断两者是否应当归为一类。因为只有对同一个组件的测试才会导致相似日志的出现，对不同组件的测试，即使产生非常相似的错误日志，也应当归属于不同组件的缺陷。此外，基于实际效果的对比，本文在具体实现时又对去重算法做出了 2 处改进：1）删除标记字段；2）将 LCS 算法的细粒度从字符级提高到基于特殊符号分割的单词级。

算法 2 错误日志去重算法

输入：$\log_{1\sim N}$：N 组错误日志
输出：G：分类集合
1. $G = \{\{1\}, \{2\}, \{3\}, \cdots, \{N\}\}$
　　//初始化每组日志的组别，一开始单独成组
2. for $i = 1; i \leqslant N; i++$ do
3. 　logstr[i] = convert2str($\log_i$)
　　//将每组日志转换为连续的字符串
4. end for
5. for $1 \leqslant i, j \leqslant N$ do　　//枚举日志对
6. 　lcs = LCScalc(logstr[i], logstr[j])
　　//计算当前日志对的最长公共子序列长度
7. 　similarity = (lcs × 2) ÷ (len(logstr[i]) + len(logstr[j]))　　//计算当前日志对的相似度
8. 　if similarity > thresholds then
9. 　　Merge G_i and G_j　　//如果相似度大于阈值，则将当前日志对合并为同一组日志
10. 　end if
11. end for

3 实验分析

本文挑选了 10 个 Android 应用构成数据集，并使用现有研究前沿的开源工具 Hwacha 和本文实现的 iFuzzer 系统对其进行模糊测试。通过对比测试数据、测试用例生成情况和日志去重表现，证明 iFuzzer 系统的有效性。

3.1 实验设置

实验环境主要分为 PC 端和移动端 2 个部分。PC 端是运行 Windows 10 操作系统、搭载 Intel Core i5-1135G7 2.40 GHz CPU 和 16 G RAM 的个人电脑。移动端是运行原生 Android 9 操作系统、搭载骁龙 800 CPU 和 2 GB RAM 的手机。PC 端和移动端之间通过 USB 数据线连接。

3.2 测试数据集设置

Hwacha 是目前相关研究中功能相对完善的开源项目，其选取了 50 个 Android 应用构成数据

集 APKFile50 进行实验[14]。本文的实验部分选取了 APKFile50 中缺陷数量最高的 10 个应用构成数据集 APKFile10 进行测试，以比较 Hwacha 和 iFuzzer 的实际表现。

3.3　缺陷发现能力分析

通过对 APKFile10 数据集中的 10 个 Android 应用进行了测试，整理得到的实验数据见表 3 所列。

表 3　APKFile10 实验数据

Tab. 3　APKFile10 experimental data

应用名称	Hwacha				iFuzzer			
	测试用例	错误	归类	实际类别	测试用例	错误	归类	实际类别
Auction	600	32	31	1	561	8	1	1
Band	559	26	20	2	524	10	2	2
HappyPoint	520	0	0	0	285	1	1	1
HiMart	550	26	26	2	516	9	6	6
IndepCapaign	660	24	20	2	566	8	1	1
JikBang	560	36	19	1	547	4	1	1
KakaoStory	559	28	8	3	493	82	8	8
KakaoTalk	546	18	18	8	524	20	3	3
Line	578	19	19	2	578	52	26	26
Naver	560	22	10	6	584	69	9	9
总计	5 692	231	171	27	5 178	263	58	58

在实验过程中，记录了 2 个测试系统为待测试应用生成的测试用例总数、发现的应用错误总数、基于各自的去重算法得到的错误类别总数，以及经过人工分析判定的实际类别总数。以应用 Auction 为例，Hwacha 为其生成了 600 个测试用例，共计发现了 32 个错误，Hwacha 的日志去重算法判定这些错误可以分为 31 类，但是经过人工分析，判定这 32 个错误实际上属于同一类。而本系统则为 Auction 生成 560 个测试用例并发现了 8 个错误，同时日志去重算法判定这些错误属于同一类，与人工分析结果一致。总体上看，iFuzzer 为所有待测试应用生成了 5 178 个测试用例，比 Hwacha 生成的测试用例总数少约 9%；iFuzzer 总计发现了 263 个错误，比 Hwacha 发现的错误总数多约 14%。

对测试发现的所有错误进行人工分析，得到具体的错误种类分布如图 6 所示。对 10 个应用进行测试而发现的 58 类错误中，有 45 类可以明确归类于 6 种错误大类中，还有 13 类暂时无法通过日志信息归类，需要结合逆向和调试技术进行深入分析来确定类别。其中，NullPointerException，即空指针错误出现的次数最多，共计 32 次。此类错误出现的原因主要是目标组件在使用 Intent 中的数据之前没有判断值是否为空。其次是 ClassNotFoundException，这类错误出现的原因主要是因为开发人员在 AndroidManifest.xml 文件中静态注册了某个组件，但是并没有编写对应的 Java 代码，或者是因为某个组件的源代码的版本迭代而被删除，但是其在 AndroidManifest.xml 文件中的静态注册代码却没有随之删除。

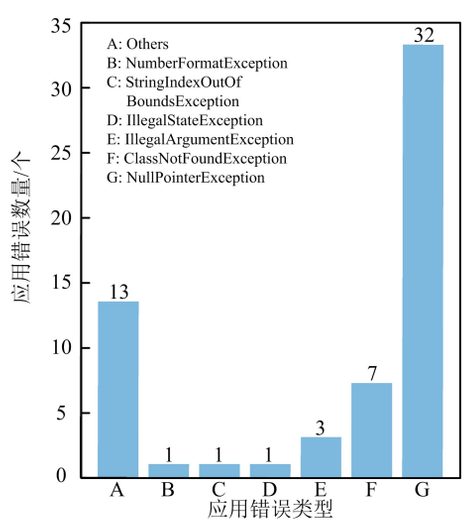

图 6　应用错误种类分布统计图

Fig. 6　Distribution statistics chart of error categories in application

3.4 生成算法效率分析

本系统和 Hwacha 的测试用例生成算法不能指定生成的测试用例的总量，只能指定为单个组件生成的测试用例最大数目。为保证实验的公平性，通过调整参数，尽量使得本系统和 Hwacha 为同一应用生成的测试用例大致相等，且大多处于 500～600 之间。需要注意的是，本系统为 HappyPoint 仅仅生成了 285 个测试用例，而且不能通过调整参数继续增加。这是因为本系统使用了枚举式的生成算法，所能生成的测试用例总数存在上限。

通过比较 4 类指标的统计数据，可以观察到本系统虽然生成的测试用例总数更少，但是触发的应用错误总数更多，而且这些错误也对应着更多类别的真实应用缺陷，足以证明本系统具备更加强大的应用缺陷探测能力，这在很大程度上归功于本系统的测试用例生成算法。该算法依据 Intent 范式所包含的信息，为 Action、Category、Extra 数据分别设计了合理的生成策略，使得测试用例质量更高。

同时，本文设计的算法还实现了合理的能量调度策略，将计算资源集中到对 Intent 范式更为丰富的应用组件的测试工作中去。通过统计本次实验中为所有组件生成的测试用例数量，绘制了如图 7 所示的散点图。散点图中的每一个点对应着一个组件，点的横坐标对应该组件的测试用例枚举空间，点的纵坐标对应着该组件实际生成的测试用例数量。借助拟合分布的曲线，可直观地观察到以 Hwacha 为组件生成的测试用例总数与组件范式对应的枚举空间的大小并没有明显的相关性，而以 iFuzzer 为组件生成的测试用例总数与组件范式对应的枚举空间的大小大致为正相关关系，能够为枚举空间较大的组件生成更多的测试用例，同时不为枚举空间较小的组件生成冗余的测试用例，这体现了本系统在生成测试用例方面具备更合理的能量分配策略。

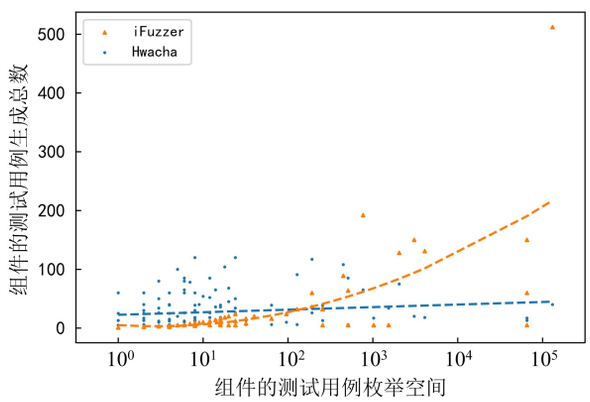

图 7 测试用例总数与测试用例枚举空间的关系对比

Fig. 7 Comparison of the relationship between the total number of test cases and the enumeration space of test cases

3.5 去重算法效果分析

为了比较 iFuzzer 和 Hwacha 各自实现的错误日志去重算法的准确性，本文使用这 2 个算法同时在本系统和 Hwacha 测试得到的 2 个错误日志数据集上进行了交叉测试。通过分析每次测试的去重归类结果，记录当前使用的去重算法将多少组日志归类错误，统计相关数据，绘制了如图 8 所示的统计图。

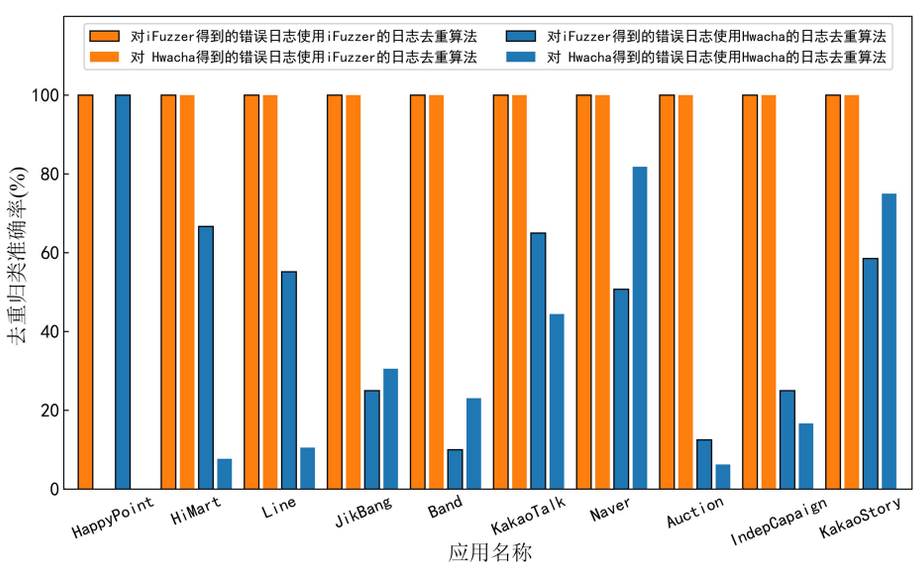

图 8 交叉测试后得到的去重归类准确率统计图

Fig. 8 Statistical chart of the deduplication accuracy after cross-testing

去重归类准确率可表示为：

$$r = 1 - w/l \tag{1}$$

式中，l 为当前的错误日志组数，w 为当前去重算法错误归类的日志组数。

从图 8 中可以直观地观察到，在当前实验中，不管是使用 iFuzzer 测试得到的错误日志数据，还是使用 Hwacha 测试得到的错误日志数据，本文提出的去重算法都能够准确将日志去重并归类，综合表现远远优于 Hwacha 提出的去重算法。（由于 Hwacha 在对应用 HappyPoint 的测试过程中没有得到错误日志，因此无法计算其去重归类准确率。）

此外，本文还统计了 iFuzzer 和 Hwacha 在实验过程中针对每个应用所触发的错误日志总数以及它们各自将这些错误日志进行归类得到的类别总数，并计算错误日志类别总数与错误日志总数的比值，绘制了如图 9 所示的统计图。从图 9 中可以看到，相较于 Hwacha，iFuzzer 归类得到的错误类别总数比错误日志总数的比值更低。因此，在面对同样数量的错误日志时，iFuzzer 输出的错误日志类别数量更低，能够进一步降低后续人工分析的工作量。同时，由表 3 可知，本系统在当前实验中的错误分类结果与人工分析结果相同，因此其去重算法的正确性也得到了证明。（由于 Hwacha 在对应用 HappyPoint 的测试过程中没有得到错误日志，因此无法计算该应用的错误日志归类总数与错误日志总数比值。）

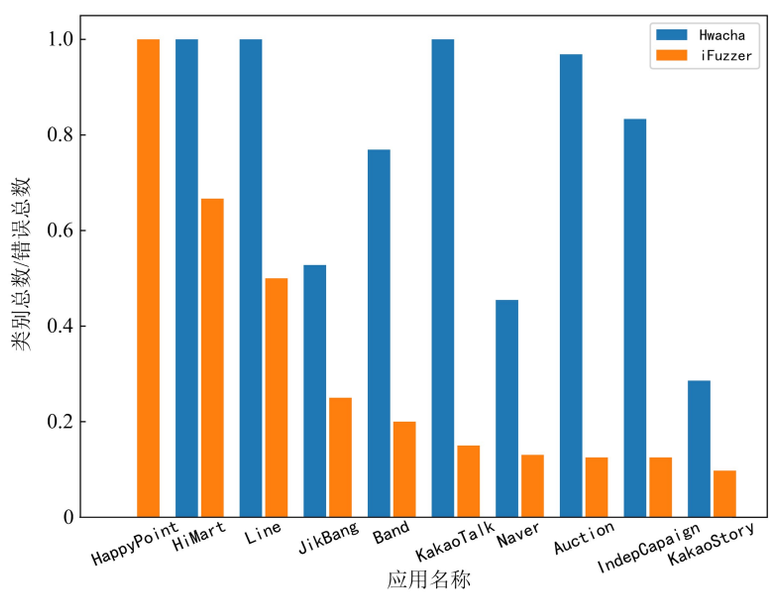

图 9　类别错误日志归类总数与错误总数比值的对比

Fig. 9　Comparison of the ratio of the total number of error log categories to the total number of errors

4　结束语

本文旨在研究面向 Android 应用的软件缺陷测试方法，并实现了一个 Android 应用组件间通信的模糊测试系统 iFuzzer。iFuzzer 能够分析 APK 源文件并提取组件信息，构建应用组件的 Intent 范式，并在此基础上生成大量的 Intent 对象进行测试，用于探测发现 Android 应用的潜在缺陷，从而降低应用程序运行异常甚至崩溃的可能性。同时，本系统还实现了错误日志去重功能，能够识别同一缺陷产生的多条错误日志并准确归类，降低了后续人工分析缺陷根源时的工作量。实验结果表明，本系统能够有效地发现 Android 应用中的软件缺陷，且在测试用例生成质量和错误日志去重归类方面表现出色。在实际应用方面，本系统可以参与到软件的测试开发工作中，辅助开发人员发现潜在的缺陷，提高应用的健壮性。

参 考 文 献

[1]　友盟+. 2021 年第一季度 U-APM 移动应用性能体验报告[EB/OL]. (2021-05-28)[2024-02-28]. http://finance. sina. com. cn/tech/2021-05-28/doc-ikmxzfmm5186339. shtml.
Umeng+. Performance experience report of U-APM mobile application in the first quarter of 2021[EB/OL]. (2021-05-28)[2024-02-28]. http://finance. sina. com. cn/tech/2021-05-28/doc-ikmxzfmm5186339.

shtml. (in Chinese)
[2] UI/Application Exerciser Monkey[EB/OL]. (2023-04-12)[2024-02-28]. https://developer.android.google.cn/studio/test/other-testing-tools/monkey.
[3] MonkeyRunner [EB/OL]. (2023-11-24)[2024-02-28]. https://developer.android.google.cn/studio/test/monkeyrunner.
[4] UI Automator[EB/OL]. (2024-01-03)[2024-02-28]. https://developer.android.google.cn/training/testing/other-components/ui-automator?hl=zh-cn.
[5] MAJI A K, ARSHAD F A, BAGCHI S, et al. An empirical study of the robustness of inter-component communication in Android[C]//Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks. [S. l.]: IEEE, 2012:1-12.
[6] YE H, CHENG S Y, ZHANG L B, et al. DroidFuzzer: fuzzing the Android APPs with IntentFilter tag [C]//Proceedings of International Conference on Advances in Mobile Computing & Multimedia. [S. l.: s. n.], 2013:68-74.
[7] YANG K, ZHUGE J W, WANG Y K, et al. IntentFuzzer: detecting capability leaks of Android applications[C]//Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. [S. l.: s. n.], 2014:531-536.
[8] 李川, 刘宝旭. ParaIntentFuzz: 安卓应用漏洞的并行化模糊测试方法[J]. 计算机工程与应用, 2018, 54(4):110-116.
LI Chuan, LIU Baoxu. ParaIntentFuzz: Android applications parallel fuzzing system[J]. Computer Engineering and Applications, 2018, 54(4): 110-116. (in Chinese)
[9] KWANGHOON C, MYUNGPIL K, BYEONG-MO C. A practical intent fuzzing tool for Robustness of Inter-component communication in Android APPs[J]. Transactions on Internet and Information Systems, 2018, 12(9):4248-4270.
[10] 赵赛, 刘昊, 王雨峰, 等. Android 组件间通信的模糊测试方法[J]. 计算机科学, 2020, 47(11A):303-309.
ZHAO Sai, LIU Hao, WANG Yufeng, et al. Fuzz testing of Android inter-component communication[J]. Computer Science, 2020, 47(11A):303-309. (in Chinese)
[11] JIANG Z Y, JIANG X Y, HAZIMEH A, et al. Igor: crash deduplication through root-cause clustering [C]//Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l.]: ACM, 2021: 3318-3336.
[12] Skylot/jadx[EB/OL]. (2023-04-02)[2024-02-28]. https://github.com/skylot/jadx.
[13] Content Provider[EB/OL]. (2023-06-07)[2024-02-28]. https://developer.android.google.cn/reference/android/content/ContentProvider?hl=en.
[14] MILLER B P, FREDRIKSEN L, SO B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12):32-44.

作 者 简 介

李 阳
男,1990 年生,副教授,研究方向为软件与系统安全
E-mail:liyanghf@nudt.edu.cn

文廷科
男,2000 年生,硕士研究生,研究方向为软件与系统安全
E-mail:wentk@nudt.edu.cn

马慧敏
女,1980 年生,副教授,研究方向为网络空间安全
E-mail:mahuimin17@nudt.edu.cn

王瑞鹏
男,1997 年生,博士研究生,研究方向为软件与系统安全
E-mail:wangruipeng@nudt.edu.cn

李倩玉
女,1995 年生,博士研究生,研究方向为网络空间安全
E-mail:liqianyu@nudt.edu.cn

潘祖烈
男,1976 年生,博士,教授,博士研究生导师,研究方向为软件与系统安全
E-mail:panzulie17@nudt.edu.cn

责任编辑 殷文卓