计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200100122

Android 组件间通信的模糊测试方法

赵 赛[1] 刘 昊[1] 王雨峰[1] 苏 航[1] 燕季薇[2,3]
1 北京工业大学信息学部 北京 100124
2 中国科学院软件研究所软件工程技术研发中心 北京 100190
3 中国科学院大学 北京 100190

摘 要 Android 操作系统提供了丰富的应用程序间消息传递机制,其中基于意图的通信是 Android 应用程序组件间的一种重要通信机制。该机制促进了应用程序间的协作,并通过增加组件重用减轻了开发人员的负担。但是这一消息传递机制可能被滥用,例如应用程序将错误消息发送给目标应用程序,从而导致目标应用程序崩溃。针对这个问题,提出一种基于模糊测试的健壮性检测方法,并实现了意图模糊测试工具 ICCDroidFuzzer。该方法通过静态分析获取组件相关信息来构造测试套件,并将其发送给目标组件,同时监测 Android 系统日志,以发现是否存在运行时崩溃。使用 ICCDroidFuzzer 检测了 420 个真实的商业应用程序,通过对实验结果进行分析,发现了 19 种导致应用程序崩溃的异常。该工具可以自动化地对应用程序的健壮性进行测试,适用于没有人为干预的大量 Android 应用程序的测试。

关键词:组件间通信;意图;模糊测试;健壮性
中图法分类号 TP311.5

Fuzz Testing of Android Inter-component Communication

ZHAO Sai[1],LIU Hao[1],WANG Yu-feng[1],SU Hang[1] and YAN Ji-wei[2,3]
1 Department of Informatics,Beijing University of Technology,Beijing 100124,China
2 Technology Center of Software Engineering,Institute of Software,Chinese Academy of Sciences,Beijing 100190,China
3 University of Chinese Academy of Sciences,Beijing 100190,China

Abstract The Android operating system provides a rich inter-application messaging mechanism,in which intent-based communication is an important inter-component communication mechanism in Android. This mechanism facilitates the collaboration of applications and reduces the burdens for developers through increasing component reuse. It is possible that this message-passing mechanism will be abused,such as the application send erroneous messages to the target application,which can result in the target crash. Aiming at this problem,a robustness detection method based on the fuzzy test is proposed and an intent fuzzy test tool ICCDroidFuzzer is implemented. The method uses static analysis to obtain component-related information to construct the test suites and send them to the target components. At the same time,the tool monitors the Android system logs to find if there is a run crash. We examined 420 real business applications using ICCDroidFuzzer. The results demonstrate 19 exceptions that cause the application crash. This tool automatically tests the robustness of applications and is suitable for testing a large number of Android applications without human intervention.

Keywords Inter-component communication,Intent,Fuzzy test,Robustness

1 引言

随着科学水平和社会经济的迅速发展,智能手机在日常生活中扮演着越来越重要的角色。据统计,截止到 2019 年第 4 季度,运行 Android 系统的设备占据全球智能手机 87% 的市场份额[1],其中在 Android 系统上运行的应用程序仅 Google Play 中就有 280 万个[2]。由于 Android 手机系统的开放性,以及 Google Play 上没有采取严格的安全审核,获取官方开发者签名比较简单等原因,Android 系统在迅速发展的同时也暴露了许多问题。

在 Android 的众多问题研究中,应用程序组件间通信过程存在的缺陷日益受到关注。Android 提供消息传递机制来实现组件间通信,简称 ICC(Inter-Component Communication)。这种通信方式在日常生活中有着很多的应用场景。例如,一个音乐类应用程序可以将某一首歌曲分享到聊天类应用程序;一个电子支付类应用程序可以被多个第三方应用程序调用用于支付;一个餐厅评价类应用程序可以请求地图类应用程序提供餐厅位置等等。Android 开发人员可以利用这种通信机制减轻负担并促进功能重用,同时利用其他应用程序提供的现有数据和服务给用户提供无缝的应用程序体验。

但是,这种通信机制将应用程序组件暴露在外部输入下,任何应用程序都可以调用该组件,包括恶意的应用程序也可

本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61672505)
This work was supported by the National Natural Science Foundation of China (61672505).
通信作者:赵赛(zhaosai94@emails.bjut.edu.cn)

以携带一些恶意信息发送给该组件。例如,有的应用程序分享一些开发者无法处理的信息到某应用程序,从而破坏程序稳定性等。对于暴露的组件而言,外部的输入源包括触摸屏、麦克风和不可信的第三方应用程序等。以视频类应用程序为例,这些应用程序可以播放常见的视频格式。然而,这些应用程序的健壮性是不同的,在面对不同格式的输入视频时,应用程序可能会崩溃或者无法加载。在 Android 上下文中,这些崩溃在堆栈跟踪中表现为各种异常。

目前已有的工作中,有针对权限、隐私数据泄露以及组件劫持的检测技术[22-24],也有检测应用程序组件通信健壮性的工作[29-30]。但是,已有检测应用程序健壮性的工作对于测试用例的构造都仅考虑了随机值和 xml 文件中的信息,而忽略了消息中所携带的 Extras 属性值,但是向组件发送消息时 Extras 携带了一些重要的额外信息,比如发送邮件时所需的邮件主题、邮件接收人等等。因此,有必要提取程序中的 Extras 属性值来构造测试用例。

Android 应用程序组件间通信时携带的属性种类多且复杂,普通的随机探索仅可能检测到少量异常。本文提出了一种基于模糊测试的健壮性检测方法来检测 Android 应用程序对输入数据时的反应情况,该方法有助于开发者测试应用程序的健壮性,从而减少应用程序崩溃的情况。本文还实现了工具 ICCDroidFuzzer,该工具能够自动地将大量测试用例发送给所有目标组件,并且不需要知道被测试组件的源代码,可以很容易地在任何 Android 设备上进行健壮性测试。实验结果表明,该方法能有效检测出 Android 组件间通信过程中引起应用程序崩溃的异常。

2 背景知识

2.1 Android 组件间的通信机制

在 Android 应用程序中,一个组件可以使用意图(Intent)机制启动另一个组件。意图是包含目标组件地址以及其他可选数据的一种消息传递对象[3]。

Intent 分为显式和隐式两种。显式 Intent 通过提供目标应用程序的包名或完全限定的组件名称来指定哪个应用程序接收 Intent。隐式 Intent 不指明特定组件,而是声明要执行的常规操作,允许来自另一个应用程序的组件处理它。对于显式 Intent,Android 系统不用对 Intent 做任何解析,系统找到指定的目标组件,触发即可;对于隐式 Intent,系统需要解析 Intent 得出满足隐式 Intent 的条件,然后去系统中查找与之匹配的组件,如果找到符合条件的组件,就启动 Intent[4]。例如,对于一个存储了通讯录信息的应用程序,当用户点击通讯录中的某个街道地址时,通讯录应用程序需要请求地图类应用程序显示该地址。为达到这个目的,通讯录应用程序可以发送一个显式 Intent 给 Google Maps,或者发一个隐式 Intent 给任何可以提供地图功能的应用程序(如 Yahoo! Maps 或 Bing Maps),让用户来选择使用哪个应用程序。

2.2 利用 Intent 通信的组件

Android 定义了 4 种类型的组件[5],其中的 Activity,Service 和 Broadcast Receiver三大组件可以利用 Intent 进行通信。

Activity[6](活动):一个可以与用户交互的图形组件,执行拨打电话、发送电子邮件、拍摄照片等操作。Intent 可以启动 Activity(startActivity(),startActivityForResult())。

Service[7](服务):一种在后台运行的组件,执行长时间运行的操作,不提供界面。例如,当用户使用其他应用程序时,Service 会在后台播放音乐。Intent 可以启动、停止和绑定 Service 等(startService(),stopService(),bindService())。

Broadcast Receiver[8](广播接收器,以下简称 BR):利用 BR 组件,系统能够向应用程序传递消息,如通知屏幕已关闭、电池电量不足等消息。Intent 可以向 BR 发送信息等(sendBroadcast())。

2.3 Intent 机制的安全风险

Android 应用程序的组件分为私有组件和公有组件(也叫 exported components,即暴露组件)。如果一个组件是私有的,则可以与之交互的组件是来自相同应用程序的组件,或者是来自相同 UID 的另一个应用程序的组件。如果一个组件是公有的,那么其他应用程序的组件可以与其交互(启动 Service、启动 Activity 等)[9],这种组件就是暴露组件。本文研究的对象是应用程序中的暴露组件。

对于暴露组件来说,任何应用程序都可以给该组件发送 Intent,而这些应用程序中有可能存在恶意的应用程序。换言之,恶意应用程序可以通过构造某些开发者预期之外的 Intent 并发送给暴露组件,这个过程可能会造成一系列问题,如破坏程序稳定性,造成应用程序崩溃等。

2.4 模糊测试

模糊测试[10]是一种简单、有效的测试方法,用来验证程序的质量及可靠性。模糊测试已经用于评估各种操作系统[11-13]的健壮性。最早对模糊测试的使用可以追溯到 1989 年,Miller 教授开发了一个模糊器[11],用以检测 UNIX 的健壮性,其关注点并不是评价系统的安全性,而是评价系统的健壮性。

模糊测试[14]是一种通过向待测试的目标软件输入一些半随机的数据并监视异常结果来发现软件故障的方法。所谓半随机,是指对于目标程序来讲,输入数据的大部分是有效且合法的,而其他部分的数据是不满足输入格式规范的。由于目标程序在编写时可能未考虑到对所有非法数据的异常处理,因此半随机数据很有可能造成目标程序崩溃,从而触发相应的问题[15]。模糊测试检测的是被测对象接收输入数据时的健壮性。健壮性[16]是指系统的稳定性,即在异常情况发生时系统是否能够正常运行。本文利用模糊测试方法生成半随机的 Intent,检测应用程序在接收到 Intent 后的健壮性。

3 针对 ICC 健壮性的模糊测试

本文研究的是 Android 组件间通信的健壮性,其中测试对象是应用程序中的暴露组件。当一个暴露组件接收到非预期的 Intent 时,检测该组件是否能够正常处理该 Intent。健壮性检测方法中最重要的是尽可能多地构造不同的 Intent 测试用例,以达到检测目的。因此,利用模糊测试的方法检测组件通信的健壮性。在构造 Intent 测试用例时,需要获取目标应用程序中的组件信息,本文利用静态程序分析的方式获得组件信息。在收集组件信息后,按照一定的策略构建测试用例。ICC 模糊测试过程如图 1 所示,主要分为 4 个模块:组件信息提取模块、测试套件构造模块、测试套件发送模块、

结果收集与分析模块。

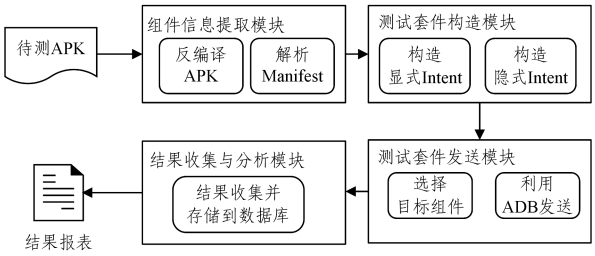

图 1　ICC 模糊测试过程

Fig. 1　ICC fuzz testing process

1）组件信息提取模块：反编译 APK，利用 XML 解析技术分析 Manifest 文件，获取 APK 中组件的详细信息；通过静态分析获取程序中组件的相关属性值。

2）测试套件构造模块：基于 1）中获取的待测组件信息和 Android 官方提供的标准常量值[17]，根据构造规则生成测试套件。

3）测试套件发送模块：根据一定的发送方法，将 Intent 测试用例发送给目标组件。

4）结果收集与分析模块：测试结束后，收集 Android 系统反馈的信息并分析。

3.1　组件信息的提取

3.1.1　静态注册的组件信息提取

AndroidManifest.xml 文件是整个 Android 应用程序的全局描述文件，不仅包含 Android 应用的名称、图标、访问权限等整体属性，还包含 Android 应用程序四大组件信息，如图 2 所示。

```
〈activity android:name="FooActivity"
  android:exported="true"
  android:permission="com.intent.permission.Foo"〉
  〈intent-filter〉
    〈action android:name="android.intent.action.EDIT"/〉
    〈category android:name="android.intent.category.DEFAULT"/〉
    〈data android:scheme="http"android:host="foo"/〉
  〈/intent-filter〉
〈/activity〉
```

图 2　FooActivity 的 Manifest 文件

Fig. 2　Manifest file of FooActivity

为了提取静态注册的组件信息，需要解析待测 APK 中的 AndroidManifest 文件。首先利用基于 DOM（Document Object Model）的 XML 分析器将一个 Manifest 文件转换成 DOM 树，然后获取 DOM 树的结点信息（即静态注册的组件信息），包括应用程序组件的 exported 属性以及应用程序组件的 Intent filter 信息。这些信息主要用于判断组件是否属于暴露组件，若 exported=true（即暴露组件），则加入待测组件的测试集。另外，在生成测试用例时，还可以利用这些组件信息生成测试用例。

3.1.2　动态注册的 BR 组件信息的提取

BR 是一个监听器，用于监听系统的广播消息，所以 BR 可以对自己感兴趣的广播进行注册。注册 BR 的方式有两种：

1）在 Manifest 中注册，这种方式被称为静态注册，3.1.1 节已提取了该方式注册的组件；

2）在代码中注册，这种方式被称为动态注册，如图 3 所示，动态注册的 NetworkChangeRec 继承自 BroadcastReceiver。

```
1. public class exampActivity extends MainActivity{
2.    private IntentFilteri Filter;
3.    private NetworkChangeRec networkChangeRec;
4.    protected void onCreate(Bundle savedInstanceState){
5.       ……
6.       iFilter= new IntentFilter();
7.       iFilter.addAction("android.net.conn.CONNECT_CHANGE");
8.       networkChangeRec= new NetworkChangeRec();
9.       registerReceiver(networkChangeRec,iFilter);
10.   }
11.   ……
12.   class NetworkChangeRec extends BroadcastReceiver{
13.      public void onReceive(Context c,Intenti){
14.         Toast.makeText(c,"change",Toast.LENGTH_LONG).show();
15.      }
16.   }
17.}
```

图 3　动态注册的 BR 示例

Fig. 3　Example of dynamically registered BR

动态注册的 BR 组件必须要在程序中调用 registerReceiver() 方法，才能实现该种组件的功能。本文首先利用 Soot[18] 对程序进行静态分析，保存程序中通过哪些语句到达了 registerReceiver() 方法；然后提取出该方法所绑定的 BR 和 Intent filter 对象；最后根据 Intent filter 对象，在所保存的语句中提取该 Intent filter 的具体信息，从而确定动态注册的广播组件名称以及所对应的 Intent filter 信息。

3.2　测试套件的构造

Intent 的构造须考虑可以触发通信中的更多崩溃异常，因此本文提供一个高覆盖率的 Intent 测试用例生成方法。

Intent 对象包含 Component name，Action，Category，Data，Type，Extras，Flags 这 7 种属性。其中，Component name 属性用于指定被启动组件的名称；Action 代表 Intent 要完成的一个抽象动作；Category 用于为 Action 增加额外的附加类别信息；Data 属性用于向 Action 属性提供操作的数据；Extras 是键值对，用于向 Intent 的组件传递附加信息。因为要将测试用例发送给目标组件，所以 Component name 属性是必不可少的，而 Action，Data，Category 和 Extras 能够体现通信过程的动作、与动作相关的数据、类型和附加信息，所以这 4 个属性也是必需的。

综合以上，本文选取 Component name，Action（a），Category（c），Data（d）和 Extras（e）属性，为每一个属性构造候选测试集。在构造测试用例时，要构造半随机的用例数据，其中包括 3.1 节中所提取的 APK 中组件相关的属性值、随机生成的属性值、Android 官方提供的标准属性值以及空的属性值。因为应用程序中可能会对这些属性值做一些条件判断后，才会继续执行下边的分支路径，所以为了覆盖到更多的代码块，须构造除所有属性值都为空或者非空之外的情况。因此，按照如下模版分类构造测试用例。

模板 1　Action，Category，Data，Extras 4 个属性值全部为非空；

模板 2　Action，Category，Data，Extras 4 个属性值有 1 个为空；

模板 3 Action, Category, Data, Extras 4 个属性值有 2 个为空；

模板 4 Action, Category, Data, Extras 4 个属性值有 3 个为空；

模板 5 Action, Category, Data, Extras 4 个属性值全部为空。

3.2.1 Extras 的分析和构造

Extras 是发送 Intent 时所携带的键值对，用来向组件传递一些额外的信息。例如，在发送电子邮件时，可以使用 KEY:EXTRA_EMAI 指定收件人，使用 KEY: EXTRA_SUBJECT 指定邮件主题等。其中，KEY 可以是 String, int 或者 boolean 类型，VALUE 可以是任何 Java 基本类型或者类。可见，Extras 可以传递丰富的信息，但这些信息并没有在 Manifest 中描述，所以有必要提取程序中的 Extras 来造测试用例，如图 4 所示。

```
1. public class HtmlActivity extends BaseActivity {
2.   ……
3.   private static final String TITLE="argtitle";
4.   private static final String MARK="argmark";
5.   ……
6.   void start(BaseActivity activity,String title,boolean mark) {
7.     Intent intent=new Intent(activity,HtmlActivity.class);
8.     intent.putExtra(TITLE,title);
9.     intent.putExtra(MARK,mark);
10.    activity.startActivity(intent);
11.  }
12.  void onCreate(Bundle savedInstanceState) {
13.    ……
14.    String title=getIntent().getStringExtra(TITLE);
15.    boolean mark=getIntent().getBooleanExtra(MARK,false);
16.    ……
17.  }
18.}
```

图 4 Extras 示例

Fig. 4 Example of Extras

可以使用 putExtra()方法添加 Extras 数据，该方法接收参数 KEY 和 VALUE，如图 4 第 8,9 行所示；还可以创建一个包含所有 Extras 数据的 Bundle 对象(Bundle：从 Stringkey 到各种 value 的映射)，然后使用 putExtra()将 Bundle 插入 Intent 中；也可以使用各种 getExtra()方法获取 Extras 数据，如图 4 第 14,15 行所示。

为了提取程序中的 Extras，我们定位 getStringExtra，getIntExtra 和 getBooleanExtra 这些 API 的调用，并获取 API 中相关组件的 key 和 value 并保存下来。由于 Bundle 类型的数据无法通过 ADB 命令发送，因此本文主要针对 String, int 和 boolean 3 种基本类型进行了变化。所构造的 Extras 中，KEY 值是上述所提取的 key，VALUE 值包括上述提取的 value，空和其他，其中随机字符串的长度范围是 20～50 字节。具体的候选测试集如下所示：

StringExtras：{KEY：key；VALUE：value，null，randomString}

IntExtras：{KEY：key；VALUE：value，null，randomint}

BooleanExtras：{KEY：key；VALUE：null，true，false}

3.2.2 显式 Intent 的构造

显式 Intent 须指定目标组件的名称，其他参数及其各参数的候选测试集如下所示：

a：{标准 Action 常量，随机字符串}

c：{标准 Category 常量，随机字符串}

d：{随机 URI}

e：{StringExtras，IntExtras，BooleanExtras，随机键值对}

标准 Action 和 Category 常量值是指 Android 官方提供的标准常量值，虽然开发者也可以构造一些自己的常量值，但是这些标准的常量值是大家公认的，所以有必要加入候选测试集。

随机 URI 须按照“scheme://host:port/path”格式构造。其中，scheme 用于指定数据的协议，如 http；host 用于指定数据的主机名；port 用于指定数据的端口；path 用于指定数据的路径。这些候选测试集中随机 URI 的长度范围为 20～50 字节，Action 和 Category 的随机字符串长度范围为 0～15 字节。

3.2.3 隐式 Intent 的构造

当隐式 Intent 可以与 Intent filter 声明的 Action，Data 和 Category 匹配时，系统才会将该 Intent 传递给目标组件。Android 系统将 Intent 的内容与在目标应用程序的 Manifest 文件中声明的 Intent filter 进行比较，从而找到要启动的相应组件。如果 Intent 与多个 Intent filter 兼容，则系统会显示一个对话框，支持用户选取要使用的应用。因此，使用 Manifest 中的 Intent filter 信息构造隐式 Intent。隐式 Intent 测试用例的各个属性候选测试集如下：

a：{标准 Action 常量，随机字符串和 Intent filter 中的 Action 信息}

c：{标准 Category 常量，随机字符串和 Intent filter 中的 Category 信息}

d：{随机 URI 和 Intent filter 中的 Data 信息}，须确认 Intent filter 中是否有 scheme, host, port, path，如果有，则按照“scheme://host:port/path”的格式构造 Data 属性值。

e：{StringExtras，IntExtras，BooleanExtras，随机键值对}

3.3 测试套件的发送

利用 ADB 命令[19]给目标应用程序组件发送所构造的 Intent。构造的 ADB 命令示例如下：

```
adb shell am start
-n com.duowan.kiwi/com.duowan.EntryActivity
-a android.intent.action.MAIN
-c android.intent.category.DEFAULT
-d http://www.ford.com/
-es test abc
```

根据 ADB 命令的语法，前缀“adb shell am start”表示 Android 启动由 *-n* 命名的组件，选项 *-a* 表示操作名称，*-c* 表示 category，*-d* 表示某些数据的 URI，*-es* 表示附加信息。

3.4 日志的获取

组件接收到 Intent 后可能会被正常唤起或者发生崩溃，为了收集并统计组件产生的异常信息，本文监控 Android 移动设备上的日志，实时捕获所有的记录(主要记录的是 Android 操作系统、应用程序和 Intent 测试用例的日志)。因此，需要收集的数据包括：

1)系统和应用程序的日志，包括一般级、警告级和错误级的日志；

2）实验过程中测试用例的信息。

Logcat[20]是 Android 系统中的一个命令行工具，可用于获取系统日志信息。除此之外，可以使用 Eventlog 收集更全面的异常信息。Eventlog[21]用于记录系统级事件，例如垃圾回收事件、活动管理器状态等。因此，需要利用 Eventlog 对组件崩溃的系统级日志进行收集。

4　实验结果分析

本研究的实验环境是 Android 7.1.2 版本的 MiA1，具体参数如表 1 所列。

表 1　测试设备的详细参数

Table 1　Details of testing device

设备名称	系统版本	内核版本	处理器
MiA1	7.1.2	3.18.31	ARMCortex-A53

对豌豆荚上 14 个分类的每个分类选取受欢迎的前 30 个 Android 应用程序，利用基于模糊测试的自动化检测方法进行实验，并从以下 4 个角度对实验结果进行分析。

1）所测应用程序抛出的异常种类及数量，以及引起异常的原因；

2）在 3 种组件中，暴露组件所占的比例，以及每种组件抛出异常的次数；

3）14 类被测应用程序接收到 Intent 后发生的崩溃次数和抛出的异常数量；

4）Extras 属性值的构造对实验的影响。

4.1　异常的种类及数量统计

组件接收到 Intent 后，会因为异常等原因出现程序崩溃。异常具体分为 Checked Exception 和 Unchecked Exception。当对目标组件发送 Intent 时，若所属应用程序中没有对异常进行捕获，则会引起应用崩溃等问题。

根据实验结果分析得到被测应用程序抛出了 19 种异常，对每种异常发生的数量进行统计，如表 2 所列，其中大部分异常属于 Unchecked Exception。据统计发现其中发生频率最高的异常是 NullPointerException，该异常的发生是因为应用程序在调用某个对象的时刻，该对象被赋值为 null；其次是 ClassNotFoundException，该异常是遍历 CLASSPATH 后无法找到对应的 class 文件。可见，前两种异常所占的比例就达到了 73.1%，它们都是由于没有判断所接收的对象是否存在导致的。因此，开发人员在开发过程中，应当对接收的 Intent 进行分析，判断所需要的数据是否存在，从而避免此类异常的发生。

表 2　异常的种类及数量

Table 2　Type and number of exceptions

异常种类	数量	所占比例/%
java.lang.NullPointerException*	182	54.3
java.lang.ClassNotFoundException+	63	18.8
java.lang.IllegalStateException*	22	6.7
java.lang.NumberFormatException*	19	5.7
java.lang.RuntimeException*	16	4.8
java.lang.IllegalArgumentException*	6	1.8
java.lang.ArrayIndexOutOfBoundsException*	4	1.3
java.lang.StringIndexOutOfBoundsException*	3	0.9
android.content.res.Resources$NotFoundException*	3	0.9
kotlin.NotImplementedError−	2	0.6
java.lang.UnsupportedOperationException*	2	0.6
java.lang.UnsatisfiedLinkError−	2	0.6
java.lang.SecurityException*	2	0.6
java.lang.InstantiationException+	2	0.6
android.view.WindowManager$BadTokenException*	2	0.6
android.support.v4.app.ao+	2	0.6
java.lang.IndeOutOfBoundsException*	1	0.2
java.lang.ClassCastException*	1	0.2
java.io.NotSerializableException+	1	0.2

注：* 表示 UncheckedException，+表示 CheckedException，−表示 Error

4.2　3 种组件发生的异常统计

本文分析了 Android 中 3 种组件（Activity，Service，BR）分别发生异常的数量与总异常数量的占比。Activity 发生的异常所占比例达到了 86%，而 Service 和 BR 分别占了 7%。

图 5 给出了 3 种组件中发生异常组件数量与暴露组件数量的比例。据统计，Activity 暴露组件占比为 13.6%；Service 暴露组件占比为 5.7%；BR 暴露组件占比为 3.9%。由此得知，暴露组件的数量和所抛出异常的数量成正比。其中，Activity 暴露组件数量最多，并且抛出异常的数量也是最多的，表明异常大部分发生在 Activity 组件中。这是一个值得关注的现象，因为 Activity 是 Android 应用中负责用户交互以及前端显示的组件。一旦 Activity 出现崩溃等现象，将严重影响应用程序的用户体验。

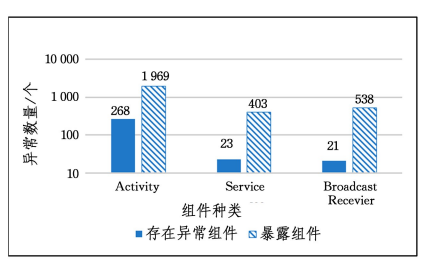

图 5　存在异常组件的占比

Fig. 5　Proportion of component exception

4.3　不同类别的应用程序的异常统计

本文按照应用程序所属类别划分，并针对每种类别的应用程序统计了引起崩溃的异常数量，如图 6 所示。其中，通讯社交、网上购物、学习考试、系统工具和新闻阅读发生的异常

较多。在这几类应用中，发生比例较大的是 NullPointerException 和 ClassNotFoundException，与 4.1 节中所分析的占比较大的两种异常一致。这些类别的应用程序在生活中极为常用，因此开发人员需多加注意。

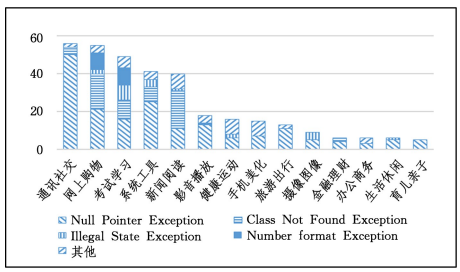

图 6 每种类别应用程序的异常统计

Fig. 6 Statistics of each category applications exception

同时，本文人工检查了所有发生异常的组件，以查看何种功能的组件易发生异常。经过人工分析发现，微博分享功能类和微信分享功能类的 Activity 组件发生的异常最多，视频播放类和 QQ 分享类的 Activity 组件其次。这几种功能是现今移动设备上必不可少的。如果开发人员不重视这些功能组件所发生的异常，应用程序的可用性则会受到影响，甚至用户可能会放弃该应用程序，转而选择其他应用程序。

4.4 Extras 取值对实验的影响

Extras 域是 Intent 的附加数据域，大部分研究在构造 Intent 时会加入该属性，但从未有工作研究 Extras 的取值对实验结果的影响程度。本文在 420 个应用中共提取了 20944 个 Extras，3 种类型的具体数量如表 3 所列。

表 3 Extras 各类型数量的统计

Table 3 Statistics on types of Extras

Extras 的类型	数量
String	14132
int	4128
boolean	2684

通过对实验结果进行分析，发现 Extras 取值为空或随机值时，每个类别应用程序崩溃的次数变化不大，如图 7 所示；并且在实验中 Extras 取值为空时未发生崩溃的应用程序，在 Extras 取随机值时依然未崩溃。

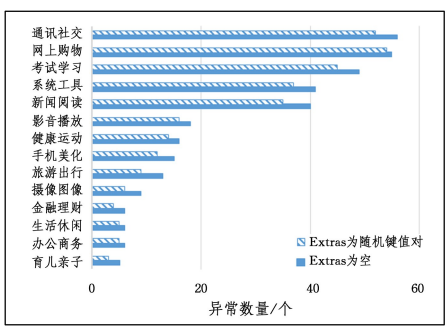

图 7 Extras 取值对实验结果的影响

Fig. 7 Effect of Extras value on experiment

同时我们发现，在这些应用程序中，有 693 个组件能够接收 Extras，通过构造带有 Extras 属性值的测试用例触发了 11 种异常，其中所构造的 StringExtras 触发的异常数量最多，如图 8 所示。

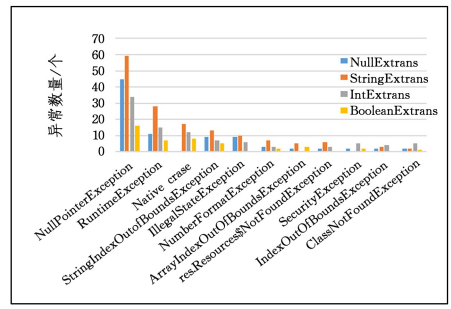

图 8 Extras 的构造对实验结果的影响

Fig. 8 Effect of the Extras construction on experiment

综上，大量的错误是由于没有对空字符串进行检查导致的，而且 Extras 域只采用随机字符串的效果并不好。所以测试时除了考虑随机字符串外，还要构造空的 Extras，以及考虑提取程序中的 Extras 属性值进行构造。

4.5 模板构造的合理性分析

本文在利用模糊测试的方法构造测试用例时，选用的是基于模板的方法，并且在构造的过程中考虑了半随机属性值的构造方式，即在构造测试用例时，要构造半随机的测试用例，其中包括 Android 官方提供的标准值、静态分析 APK 所得到的相关组件的属性值、随机生成的属性值以及空的属性值。半随机属性值的构造结果如图 9 所示，从中可以看出 3.2 节测试用例的构造是合理的。

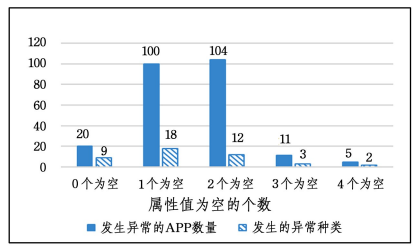

图 9 半随机属性值的构造

Fig. 9 Semi-random attribute value construction

4.6 NullPointerException 示例分析

由于根据本文的模糊测试方法所触发的异常中 NullPointerException 所占比例最高，因此对这种异常进行深入分析。为了分析引起崩溃的根本原因，本文对 SyncOrg(一个记录并整理笔记的应用)的源码进行分析，部分代码如图 10 所示。

```
1. public void onCreate(Bundle savedInstanceState) {
2.     ……
3.   // Get the intent, verify the action and get the query
4.   Intent intent=getIntent();
5.   if (Intent. ACTION_SEARCH. equals(intent. getAction())) {
6.     String query=intent. getStringExtra(SearchManager. QUERY);
7.     doSearch(query);
8.   }
9.     ……
10. }
11. private void doSearch(String query) {
12.   TextView noResultText=(TextView) findViewById(R. id. no_result_
      text);
13.   Cursor result=OrgProviderUtils. search("%"+query. trim()+"%",
      getContentResolver());
14. }
```

图 10 SyncOrg 的部分源码

Fig. 10 Part of the source code of SyncOrg

当 Intent 获取到 Extras 并赋值给 query 后，未判断 query 是否为空，导致调用 doSearch(String query)方法时抛出空指针异常，引起程序崩溃。

5　相关工作

目前已存在许多针对 Android 应用程序组件间通信的相关检测技术，有针对隐私泄露、权限泄露等问题的研究和分析。Fu 等[22]提出了一种静态缺陷检测方法，检测了隐式 Intent 可能引发的组件劫持、信息泄露和权限泄露。Li 等[23]通过静态污染分析技术跟踪从 source 到 sink 的任何敏感数据流来发现组件间隐私泄漏，并实现了检测工具 IccTA。Bohluli 等[24]实现了 IIFDroid——一种组件间信息流控制静态分析工具，用于检测 Android 应用程序中各种形式的流所产生的信息泄漏。Kun 等[25]提出了一种动态意图模糊机制来发现应用程序的权限泄露，并实现了工具 IntentFuzzer。Liu[26]基于 Intent 对 Android 应用程序的安全性进行了研究，通过研究所收集的大量应用程序的权限和 Intent，提出了利用 Intent 检测恶意应用的方法。

还有一部分针对 Android 组件间通信健壮性检测的工作。Wang 等[27]提出一种系统化测试暴露 Activity 的方法，并根据该方法生成了一组代理应用作为测试驱动来启动应用程序中的暴露 Activity 组件，但仅测试了 Activity 一种组件；而本文考虑了所有可以利用 Intent 进行通信的组件，包括动态注册的广播组件。Wang 等[28]针对组件暴露的问题，提出了一种结合 Fuzzing 技术和逆向分析的漏洞挖掘方法，并实现了工具 KMDroid，但其发现的崩溃异常种类只有 8 种，而本文发现了 19 种崩溃异常，数量远超过 KMDroid。Zhang 等[29]提出了一种基于模糊测试方法的组件通信测试方案，并实现了工具 FuzzerAPP，对常用应用程序进行鲁棒性测试，但对异常的分类并不准确；而本文对 Unchecked Exception 和 Checked Exception 的分类是准确的，并指出各分类异常带来的危害。Choi 等[30]设计了意图规范语言来描述意图的结构，并提出了一种基于 LCS 算法的故障自动分类计数方法，实现了 Hwacha 的意图模糊测试工具，但其发现的崩溃异常仅 6 种，对比表明本文能够挖掘出更多的崩溃异常。

结束语　本文提出了一种针对 Android 应用程序组件间通信健壮性的模糊测试方法。该方法使用 Android 的标准常量值、随机字符串、Manifest 中的信息以及代码中动态注册组件的相关信息来构造测试用例，并把测试用例发送给目标组件，最后收集实验数据并进行分析。最后，基于上述方法实现了全自动测试工具 ICCDroidFuzzer，完成了对 420 个应用程序的测试，挖掘出 19 种引起应用程序崩溃的异常，并且对源码分析得到抛出异常的根本原因。

通过对实验结果分析发现，Extras 取值为空或随机值时程序崩溃次数变化不大，大部分错误是由于没有对空字符串检查。因此，建议在测试时尽量优先选用空字符串构造测试用例，同时考虑提取程序中的 Extras 属性值进行构造。

未来将研究如何更好地利用反馈信息构造 Intent，以达到提高测试用例覆盖率的目的；以及如何自动化探究崩溃发生的根本原因，并定位异常发生的位置。

参 考 文 献

[1] IDC 2019[EB/OL]. https://www.idc.com/promo/smartphone-market-share/os.

[2] Google Play Store: number of apps 2019 | Statista [EB/OL]. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[3] Intent [EB/OL]. https://www.hahack.com/wiki/android-intent.html.

[4] Intent and Intentfilters [EB/OL]. https://developer.android.com/guide/components/intents-filters.html.

[5] Android components fundamental [EB/OL]. https://developer.android.com/guide/components/fundamentals.

[6] Android Activity[EB/OL]. https://developer.android.com/guide/components/activities.

[7] Android Service [EB/OL]. https://developer.android.com/guide/components/services.html.

[8] AndroidBroadcastReceiver[EB/OL]. https://developer.android.com/reference/android/content/BroadcastReceiver.html.

[9] Component Security and Permissions [EB/OL]. https://www.oreilly.com/library/view/application-security-for/9781449322250/ch04.html.

[10] SUTTON M, GREENE A, AMINI P. Fuzzingbruteforce vulnerability discovery[M]. Beijing: China Machine Press, 2009: 13-14.

[11] 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.

[12] MILLER B P, KOSKI D, LEEC P, et al. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services[R]. University of Wisconsin-Madison Department of Computer Sciences, 1995.

[13] FORRESTER J E, MILLER B P. An empirical study of the robustness of Windows NT applications using random testing [C]// Proceedings of the 4th USENIX Windows System Symposium. 2000, 4: 59-68.

[14] ZHANG X, LI Z J. Survey of fuzz testing technology[J]. Computer Science, 2016, 43(5): 1-8, 26.

[15] CHEN C, CUI B, MA J, et al. A systematic review of fuzzing techniques[J]. Computers & Security, 2018, 75: 118-137.

[16] BERTSIMAS D, SIM M. The price of robustness[J]. Operations research, 2004, 52(1): 35-53.

[17] Android StandardActionand Category [EB/OL]. https://developer.android.com/reference/android/content/Intent.

[18] Soot[EB/OL]. http://www.bodden.de/2008/09/22/soot-intra.

[19] Android ADB [EB/OL]. https://developer.android.com/studio/command-line/adb.

[20] Android Logcat [EB/OL]. https://developer.android.com/studio/command-line/logcat.

[21] Android Eventlog [EB/OL]. https://developer.android.com/reference/android/util/EventLog.

[22] FU J M, LI P W, YI Q, et al. A static detectionof security defects between inter-components communication[J]. J. Huazhong Univ. of Sci. & Tech. (Natural Science Edition), 2013, 41(S2): 259-264.

[23] LI L, BARTEL A, BISSYANDÉT F, et al. Iccta: Detecting inter-component privacy leaks in android apps[C]// IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015: 280-291.

（下转第 315 页）