摘　要

Android 应用程序通过 manifest. xml 文件配置应用与组件之间的关系、组件的基本属性以及组件的消息通信接口。对配置文件的理解以及各个配置项的准确含义一般通过阅读官方文件获得。但是，在实际应用中出现的很多配置项组合在官方文件中并没有给出完整的说明，即使能够找到说明，由于自然语言存在二义性也使得对这些组合的准确理解产生了歧义。

本课题主要以 Android 配置的组合测试为研究对象，明确 Android 组件配置及安全机制。主要内容包括以下三个部分：（1）了解和熟悉 Android 的组件配置所代表的含义。（2）针对 BroadcastReceiver 和 ContentProvider 设计了若干个测试案例，并进行了测试，收集和分析测试结果，总结出一组配置规则。（3）开发了一个工具，能够检测一个程序是否与这些规则匹配，如果不匹配，则程序可能存在一些逻辑上或开发上的矛盾、安全上的一些隐患。

关键字：Android；安卓应用程序配置项；软件测试；安卓安全；检测工具

ABSTRACT

Android applications configure the relationship between applications and components, the basic properties of components and the message communication interface of components through manifest.xml files. The understanding of configuration files and the exact meaning of each configuration item are usually obtained by reading official documents. However, many combinations of configuration items appeared in practical applications have not been fully explained in official documents. Even if instructions can be found, due to ambiguity in natural language, the accurate understanding of these combinations is ambiguous.

This topic mainly takes the combination test of Android configuration as the research object, and clarifies the configuration and security mechanism of Android components. The main contents include the following three parts: (1) Understanding and familiarizing with the meaning of Android component configuration. (2) Designed several test cases for Broadcast Receiver and Content Provider, tested them, collected and analyzed the test results, and summarized a set of configuration rules. (3) A tool has been developed to detect whether a program matches these rules. If it does not, there may be some contradictions in logic or development, and some hidden dangers in security.

Key Words: Android; Android Application Configuration Items; Software Testing; Android Security; Detection Tools

目　录

1 绪 论

1.1 本课题研究的背景和意义

随着移动互联网的快速发展，4G 时代的普及，5G 时代的出现，智能手机特别是 Android 智能手机的用户日益增多，统计数据显示，2018 年国内智能手机出货 3.9 亿部，Android 占比 89.3%。Android 是一种基于 Linux 的操作系统，其拥有自由以及开放源代码的性质。目前大部分移动设备都是基于 Android 系统的，如手机、电视、平板和智能手表等，数量远远超过基于苹果公司的 iOS 系统和 Window 公司的 Windows Phone 系统的设备。随着 Android 的逐渐发展，安全问题也随之出现，并且越来越严重，有应用本身的安全问题、应用之间的联系的安全问题和应用之外的安全问题。Android 安全问题在安卓开发中也扮演者越来越重要的地位，安全问题看不见摸不着，就像空气一样，随时随地都有可能出现，一旦出现，如果没有即使修复，将会给社会经济带来更大的伤害。从 2007 年 11 月 5 日发布的第一个版本 Android 1.0 开始，每隔一段时间都会发布一个新的版本，每个更新的版本都会修复前一个版本中存在的一些 Bug，并在新版本中添加一些新的功能。

Android 应用面临许多安全问题，包括 App 重打包、修改、篡改的威胁，还包括病毒、关键信息的泄露，除此之外还有 Webview 漏洞、进程被劫持、出局在传输的过程被劫持等等。Android 将安全设计贯穿于系统架构的各个层面，包括覆盖应用程序层、应用程序框架层、Linux 内核层和虚拟机各个环节，以确保用户数据的安全、应用程序的安全和移动设备的安全。现实中，出现的问题可能更多。总的来说，Android 开发的常见安全问题包括如下几种情况：

（1） 应用程序组件开发的安全要点。可以通过设置 Activity, Service, Content Provider, Broadcast Receiver 等组件的相应的属性和相关的元素，在代码层做好安全措施。四大组件都可以通过调用隐式 Intent 意图打开，所以这些组件只要不是对外公开的，即不允许其他应用程序访问，必须将 AndroidManifest 文件的每一个 exported 属性设置为 false。对于要和外部交互的组件，需要要对传递的数据进行安全的校验，还需要添加对相应组件的访问权限。

（2） 应用权限控制。应用权限包括应用程序自定义权限和系统内置权限。Android 开发中可以通过设置应用程序的权限来保护应用程序的安全和防止恶意软件对系统的破坏。

（3） 应用程序签名。通过采用数字签名为应用程序签名，用户可以无缝升级到新的版本，可以将应用程序模块化，可以在应用程序之间以安全的方式共享代码和数据。

（4） 数据存储加密。通过对应用程序存储的数据设置密码来保护应用程序的敏感数据。

（5） 静态代码分析。通过静态代码分析工具监测应用程序的安全隐患，并对代码进行优化。

（6） 应用加固。行业内已经出现了很多的应用加固解决方案，如百度应用加固、腾讯云应用加固、360 应用加固等等。 应用加固包括防篡改、防注入、病毒扫描、防调试四个模块，

（7） 防火墙。必要时可以为 Android 设备添加防火墙，以防止远程网络攻击。

恶意软件的数量在逐年递增，仅在 2018 年就有 434.2 万个新恶意软件，如图所示。可见，Android 安全问题是目前最需要解决的问题之一。

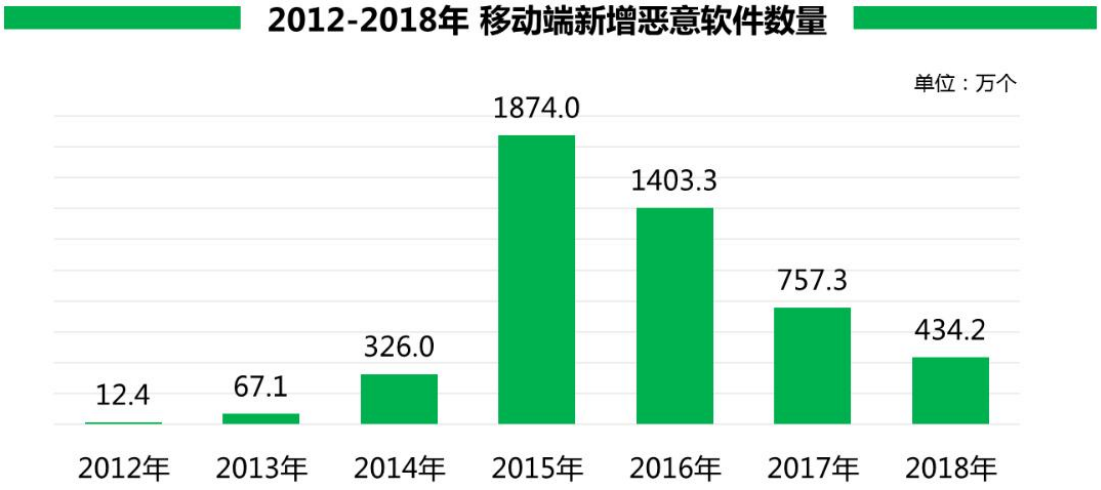

图 1-1 2012~2018 年移动端新增恶意软件数量

基于应用程序组件开发的安全要点，Android 四大基本组件在代码层面应采取的安全措施，在组件之间的通信过程中，开发者可能对权限的配置组合处理不当，导致一些安全漏洞。虽然对配置文件的理解以及各个配置项的准确含义一般通过阅读官方文件可以获得，但是，在实际应用中出现的很多配置项组合在官方文件中并没有给出完整的说明，即使能够找到说明，由于自然语言存在二义性也使得对这些组合的准确理解产生了歧义。基于此，在该课题中，通过测试的方法对典型配置组合进行测试，并对测试结果进行分析，采用自然语言的方式对配置组合的语义进行描述，为 Android 应用开发者提供语义基础。

1.2 国内外研究现状及进展

Android 应用程序的安全问题日益增长，Android 系统漏洞包括远程攻击、权限提升、信息泄露三个类别。国内外的众多学者对日益显著的 Android 应用的安全问题做了许多研究，从而对移动设备进行防护。

1.2.1 国外研究现状

Salva、Sebastien、Zafimiharisoa、Stassia R 4 人发表文章[7]，提出 Android 意图消息传递是一种将组件绑定在一起以构建移动应用程序的机制。意图是由动作和数据组成的消息种类，由组件发送到另一个组件以执行若干操作，例如，启动用户界面。意图机制简化了移动应用程序的编写，但它也可能被用作安全攻击的入口点。后者可以很容易地与组件一起发送，可以间接地将攻击转发给其他组件等等。在此背景下，本文提出了一种基于模型的安全测试方法，试图检测 Android 应用程序中的数据漏洞。换句话说，这种方法生成测试用例，以检查组件是否容易受到攻击，通过意图发送，暴露个人数据。我们的方法采用称为漏洞模式的模型正式表达的 Android 应用程序和基于意图的漏洞。然后，这是我们方法的原创性，部分规范是从配置文件和组件代码自动生成的。然后，根据漏洞模式和先前的规范自动生成测试用例。提供了一个名为 APSET 的工具，并通过对某些 Android 应用程序的实验进行评估。

William Enck、Machigar Ongtang、Patrick McDaniel 3 人发表文章[8]，通过一系列案例向我们描述了 Android 的安全模型。为了进一步检测 Android 的安全策略，他们使用 Kirin 工具从 manifest 文件中提取应用的安全策略，以确认请求的权限是否和组件分配的权限一致，通过使用运行在 Android 上的 Prolog 引擎生成符合性证明。如果应用的安全策略不符合要求，就不会安装此应用程序。他们还设计并实现了一个基于权限的恶意程序分析框架用于检测多种敏感权限组合类的程序。

1.2.2 国内研究现状

许庆富等人[1]针对如何在黑客利用 Android 程序漏洞攻击前发现潜在漏洞的漏洞检测技术研究，提出了一种基于 APK 逆向分析的应用漏洞检测技术。在逆向反编译 APK 静态代码的基础上,运用特征提取算法将 smali 静态代码解析转换为函数调用图作为特征来源，建立原始特征集合提取模型，继而通过改进 ReliefF 特征选择算法对原始特征集合进行数据降维，提取 APK 包中的漏洞特征向量，依次构建漏洞的检测规则。再结合 Android 漏洞库收录的漏洞特征对特征向量进行正则匹配，以挖掘其中潜在的安全漏洞。基于该检测方法实现了系统模型并进行对比性实验，实验结果表明此检测方法的漏洞检出率达 91%以上。因此，该漏洞检测技术能够有效挖掘 Android 应用中常见的安全漏洞。

李志杰[2]提出如果攻击者对有安全隐患的应用程序发送畸形数据，导致应用程序产生异常然后崩溃，将会造成应用拒绝服务。Android 应用程序拒绝服务漏洞不仅会降低用户体验和造成商业利益的损失，还有可能使安全类型的应用程序的防护功能被绕过而失去效果，产生更加严重的安全问题。如果是系统应用程序，拒绝服务漏洞将造成手机重启，进而会让恶意应用程序或病毒获取高级权限。目前，对于 Android 应用程序拒绝服务漏

洞的检测问题，一般采用把静态分析和模糊测试相结合的方法，但现有方案仍然存在着许多不足。

付胧玉[3]提出对 Android 应用漏洞的自动化检测问题目前主要用污点分析技术解决，其中的静态污点分析技术因为代码覆盖率高，操作过程简单得到了广泛的关注。但是，当前的静态污点分析工具基本都停留在学术研究的领域，如果这些静态污点分析工具应用到实际的漏洞挖掘工作时会产生很多问题。第一，静态污点分析工具对于硬件的要求非常高，几乎不能检测大型的 Android 应用程序。接着，在对整个程序的分析过程中会分析很多无用的数据，导致缺乏对漏洞的针对性。最后，静态污点分析技术对于分析路径是不敏感的，这样会导致大量误报。为了解决上述的问题，付胧玉等人针对传统的静态污点分析技术做了改进，设计并实现了一个新的 Android 应用漏洞检测系统 STDroid。一方面，STDroid 漏洞检测系统继承传了统静态污点分析的优点。另一方面，STDroid 漏洞检测系统增加了对系统库函数的分析封装、基于上下文选择分析目标和采用符号执行技术过滤分析结果三种改进技术。在一定程度上解决了静态污点分析技术在分析大型 Android 应用时的性能问题，提高分析过程的针对性，提高了漏洞检测的准确率。

吴丹等人[4]提出由于各种 Android 市场对应用程序审核不严格，投放到市场中的 Android 应用程序存在着大量的已知类型的安全漏洞，这些安全漏洞给用户的隐私和财产带来了严重的威胁。如何快速有效的检测出 Android 应用程序中已知类型的漏洞并定位漏洞所在代码段，成为 Android 安全领域的一个热门研究内容。他们针对 Android 应用漏洞检测技术进行了研究，对已知应用漏洞进行归纳、分析，提取应用漏洞特征信息，提出并实现了一个基于特征匹配的 Android 应用漏洞检测系统。该 Android 应用漏洞检测系统由数据库、检测脚本、Web 页面三个部分组成。经实验表明，其系统能够有效检测并定位 Android 应用中存在的已知类型漏洞。

宋丽珠等人[5]提出 Android 应用软件漏洞的存在，造成用户敏感数据被恶意窃取的矛盾日渐突出。为了更好地保护用户隐私信息，论文提出一种 Android 应用软件漏洞检测方法；首先，对不同漏洞进行归类整理分析;然后，根据不同类别漏洞特征采用相应的检测方式;最后，基于所提出的分析方法，实现了 Android 软件漏洞检测原型工具，并用该工具检测了 886 个 Android 应用软件样本，检测结果表明论文所提的方法简便有效。

张嘉元[6]提出目前智能移动操作系统被 Android 系统占领了绝大多数分量，但是 Android 系统的安全问题却令人忧虑，近年来频频爆出高危漏洞，给广大 Android 用户带来了很大的安全隐患。对 Android 系统漏洞检测进行了研究，提出了一种基于匹配的快速检测 Android 系统漏洞的方法，同时设计和实现了一个基于脚本的漏洞扫描工具。实验表明，该工具可有效检测 Android 系统漏洞，运行效果良好。

1.3 本论文研究内容

本论文主要研究 Android 配置的组合测试和基于配置的简单安全漏洞检测工具的开发。准确理解配置文件中各个配置项的含义，根据典型配置项组合，设计和实现测试案例，并收集和分析测试结果，提炼配置规则，设计和开发一个简单安全性检测工具，能够在配置层面检测 Android 应用的安全漏洞。

1.4 本论文组织结构

本论文共包含五个大模块，论文的结构划分如下：

第一章：绪论。介绍本论文的研究背景及意义，国内外研究现状和本论文研究内容以及本论文的组织结构。

第二章：Android 体系结构和 Manifest 文件。介绍了 Android 的体系结构，Manifest 文件的介绍。

第三章：应用程序的组件属性配置。介绍了静态配置 BroadcastReceiver 的若干属性、动态配置 BroadcastReceiver 的若干属性和 ContentProvider 的若干属性的配置。

第四章：检测工具的设计与实现。可行性分析、需求分析、检测工具的设计。

第五章：总结。本章对论文的内容进行了总结。

2 Android 体系结构和 Manifest 文件

2.1Android 体系结构

Android 系统结构就像一个多层蛋糕，每一层都有自己的特性和用途。其由 5 个部分组成，分别是 Applications【应用程序层】、Application Framework【应用程序框架层】、Libraries【核心类库】、Android Runtime【Android 运行时】和 Linux Kernel【Linux 内核】，具体如图 2-1 所示。

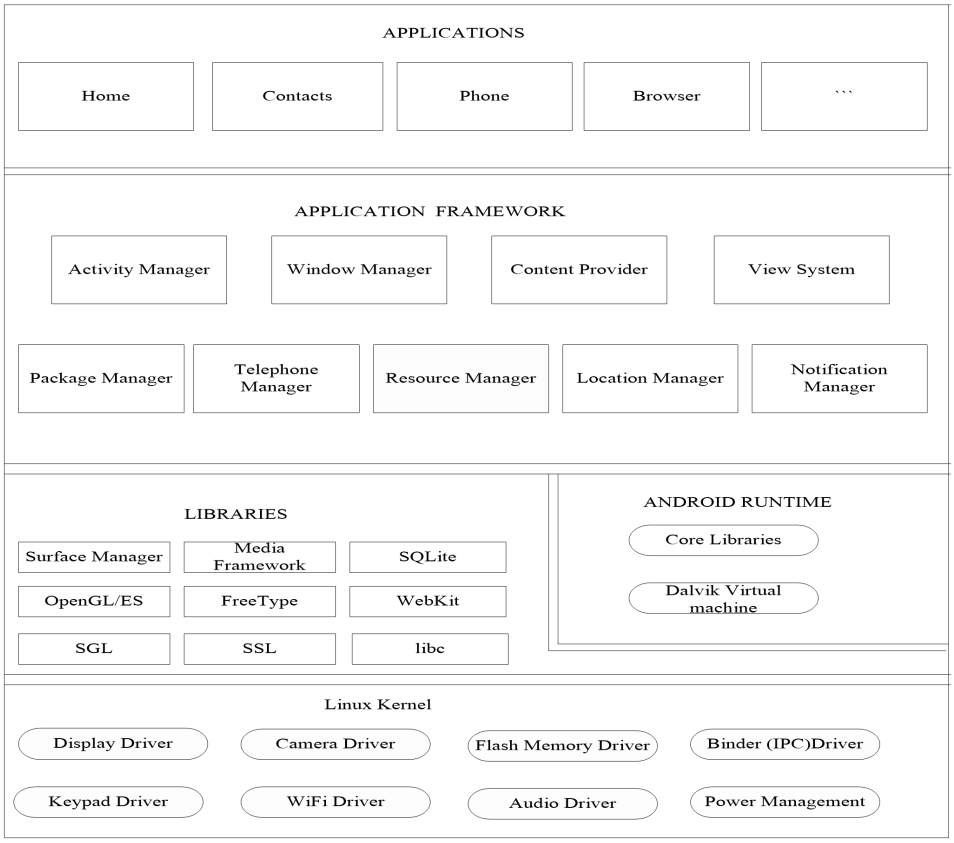

图 2-1 Android 体系结构

从图 2-1 可以看出 Android 体系结构的具体结构，接下来分别针对这几层进行分析，具体如下：

2.1.1 Applications

Android 平台装配了一个核心应用程序的集合，所有安装在手机上的应用程序都属于这一层，这些应用程序包括 SMS 程序、日历、电子邮件客户端、联系人、浏览器、地图和其他设置，所有的应用程序都是用 Java 语言写的。

2.1.2 Application Framework

应用程序框架层主要提供了构建应用程序时用到的各种 API 供开发者使用。Android 自带的一些核心应用就是使用这些 API 完成的，包括 Views【视图】、Content Provider【内容提供者】、Activity Manager【活动管理器】、Notification Manager【通知管理器】和 Resource Manager【资源管理器】等，开发者也可以通过这些 API 来构建自己的程序。

2.1.3 Libraries

核心类库中包含了系统 C 库、媒体库、界面管理、LibWebCore、SGL、3D 库、Free Type 和 SQLite 等。这些类库都是供 Android 系统的各个组件使用，一般情况都是通过应用程序框架来调用这些库，Android 应用开发者不直接调用这套库。

2.1.4 Android Runtime

Android Runtime 提供了一些核心库，提供了 Java 语言 App 中的大多数功能。另外 Android 运行时库中还包括了 Dalvik 虚拟机，Android 应用程序都运行在自己的进程上，享有 Dalvik 虚拟机为它分配的专有实例，并在该实例中执行。

2.1.5 Linux Kernel

Android 系统主要基于 Linux 内核开发，这一层为 Android 设备的各种硬件提供了底层的驱动。内核作为一个抽象层存在软件和硬件之间，强大的内存管理和进程管理，基于权限的安全模式，支持共享库，经过认证的驱动模式。

2.2Manifest 文件

每个应用程序在根目录都包含一个 AndroidManifest.xml 文件。 Manifest 文件向 Android 系统提供了应用程序的一些基本信息，以及运行该应用程序的代码所必需的系统条件。其中，Manifest 文件的以下用途尤为重要：

- 它定义了应用程序的 Java 包名称。包名称是应用程序的唯一标识。
- 它声明了构成应用程序的各个组件 — Activity、服务、广播接收器和 Content Provider。
- 它命名了每个组件的类名，并声明了组件的功能，例如，可以响应哪个 Intent 消息。通过这些声明，Android 系统可以知晓存在哪些组件、在什么场合可以启动这些组件。
- 它决定了应用程序的各个组件可以放入哪个进程中运行。
- 它声明了应用程序所必需的权限，用于访问受保护的 API 或与其他应用程序交互。
- 它还声明了其他应用程序和本程序中的组件交互所必需的权限。
- 它给出了 Instrumentation 类，当应用程序运行的时候 Instrumentation 处于开启，

Instrumentation 将在任何应用程序运行前初始化，可以通过它监测系统与应用程序之间的交互。

- 它声明了应用程序所需的 Android API 最低版本，它给出了应用程序必须链接的库。

如图 2-2 和图 2-3 是 Manifest.xml 文件的结构：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
```

图 2-2Manifest.xml 文件的结构

```
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>
        <uses-library />
    </application>
</manifest>
```

图 2-3Manifest.xml 文件的结构（续图）

2.2.1 <manifest>元素

如图 2-4 是<manifest>元素的属性：

```
<manifest  xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.woody.test"
        android:sharedUserId="string"
        android:sharedUserLabel="string resource"
        android:versionCode="integer"
        android:versionName="string"
        android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
</manifest>
```

图 2-4<manifest>元素的属性

具体信息如下：

（1）xmlns:android

xmlns 的英文是 XML namespace，即 XML 命名空间，xmlns 定义语法为 xmlns:namespace-prefix="namespaceURI"，为了解决 XML 元素和属性命名冲突。

（2）package

该属性指定本应用内 java 程序所在包的包名，它也是一个应用进程的默认名称。

（3）sharedUserId

sharedUserId 即共享用户 id，将不同应用程序的 sharedUserId 都设为相同的值，则这些应用程序之间就可以互相共享数据了。Android 给每个 APK 进程分配一个唯一的 UserID，所以是默认禁止不同 APK 访问共享数据的。若要共享数据，第一可以采用 Share Preference 方法，第二个方法就是 sharedUserId 属性。

（4）sharedUserLabel

该属性可以给共享的 userId 设置一个用户可读的标签，只有在设置了 sharedUserId 属性的前提下，这个属性才会有意义。

（5）versionCode

该属性对用户不可见，是给应用程序设置版本用的，必须是一个 interger 值代表 App 更新过多少次，第一版一般为 1，若要更新版本就设置为 2，依次论退。

（6）versionName

该属性是用户可见的，用户会通过 versionName 认识自己安装的版本号。

（7）installLocation

通过设置该属性可以决定程序的安装位置。若值等于 preferExternal，系统会优先考虑将 APK 安装到 SD 卡上，若值等于 auto，系统将会根据存储空间自己去适应，若值等于 internalOnly 是指必须安装到内部才能运行。

2.2.2 <application>元素

每一个 AndroidManifest.xml 中都有一个 Application 元素，这个元素里面声明了每一个应用程序的组件和其属性。

如图 2-5 是<application>元素的属性：

具体信息如下：

（1） android:description、android:label

这两个属性都是对该应用程序的一个介绍。这里的 android:label 属性可以设置应用程序整体的、用户可见的标签，其内容简短。android:description 属性是对该应用程序的一个详细的介绍，其内容比较长。

（2） android:enabled

android:enabled 属性可以设置当前应用程序是否能够实例化应用程序的组件。如果该属性的值为 true，并且应用程序组件的 android:enabled 属性的值表示应用程序组件可以被实例化，如果该属性的值为 false，表示所有的组件均不能被实例化。

（3） android:icon

该属性用于设置应用程序的图标。

（4） android:name

该属性的值为应用程序所实现的一个 Application 子类的完整的 Java 类名。在应用程序进程开始时，该类在所有应用程序组件中是第一个被实例化的。

```
<application  android:allowClearUserData=["true" | "false"]
              android:allowTaskReparenting=["true" | "false"]
              android:backupAgent="string"
              android:debuggable=["true" | "false"]
              android:description="string resource"
              android:enabled=["true" | "false"]
              android:hasCode=["true" | "false"]
              android:icon="drawable resource"
              android:killAfterRestore=["true" | "false"]
              android:label="string resource"
              android:manageSpaceActivity="string"
              android:name="string"
              android:permission="string"
              android:persistent=["true" | "false"]
              android:process="string"
              android:restoreAnyVersion=["true" | "false"]
              android:taskAffinity="string"
              android:theme="resource or theme" >
</application>
```

图 2-5<application>元素的属性

（5）android:permission

如果跟应用程序进行交流时，客户端需要权限，该属性用于设置权限，此刻该属性在<application>上定义，其含义是一个给应用程序的所有组件设置权限，<application>上定义的权限会被各组件设置的权限所覆盖。

（6）android:presistent

如果该属性设置为 ture，则该属性在任意时刻都保持运行状态，如果该属性设置为 false，则不保持时刻运行。默认值为 false。

（7）android:process

该属性用于设置应用程序运行的进程名，应用程序所有的组件默认情况下都运行在这个进程中，它的默认值是本应用程序 Java 类所在的包名。每个组件都可以通过设置该

属性来覆盖默认值。如果两个应用程序共享一个用户 ID 并且有相同的数字证书时，此时若设置两个应用程序的 android:process 相同，则这两个应用程序共用一个进程。

（8）android:theme

该属性可以给所有的 Activity 定义了一个默认的主题风格，每个 Activity 设置的主题风格会覆盖<application>里所定义的主题风格。

2.2.3 <activity>元素

如图 2-6 是<activity>元素的基本属性:

```
<activity
        android:enabled=["true" | "false"]
        android:exported=["true" | "false"]
        android:icon="drawable resource"
        android:label="string resource"
        android:multiprocess=["true" | "false"]
        android:name="string"
        android:permission="string"
        android:process="string" >
</activity>
```

图 2-6 <activity>元素的属性

大部分属性在前面已经叙述过，其它属性的具体信息如下：

android:multiprocess：是否允许多进程，默认是 false。该属性与 android:process 属性息息相关。如果 android:process 属性已经设置为 pro，android:multiprocess 设置为 ture，加载时，当前组件是在调用者的进程中初始化的；android:multiprocess 设置为 false，加载时，当前组件是在 pro 进程中初始化。

<intent-filter>子元素

如图 2-7 是<intent-filter>子元素的属性：

```
<intent-filter   android:icon="drawable resource"
                 android:label="string resource"
                 android:priority="integer" >
        <action />
        <category />
        <data />
</intent-filter>
```

图 2-7<intent-filter>子元素的属性

具体信息如下：

（1）intent-filter 元素的属性

android:priority：该属性的含义是执行的优先级，默认值是 0，范围是从-1000～1000，数字越大优先级别越高。

（2）action 元素

该元素是给 filter 添加一个操作。action 元素只有 android:name 一个属性。许多标准的 action 作为常量预定义在 Intent 类。例如常见的 android:name 值为 android.intent.action.MAIN，表明此 activity 是作为应用程序的入口。也可以自定义 action，注意 action 的唯一性。

（3）category 元素

该元素是给 filter 添加一个分类的名称。category 元素也只有 android:name 一个属性。许多标准的 category 作为常量预定义在 Intent 类。例如常见的 android:name 值为 android.intent.category.LAUNCHER，决定应用程序是否显示在程序列表里。也可以自定义分类，注意 category 的唯一性。

（4）data 元素

如图 2-8 是 data 元素的属性：

```
<data   android:host="string"
        android:mimeType="string"
        android:path="string"
        android:pathPattern="string"
        android:pathPrefix="string"
        android:port="string"
        android:scheme="string"/>
```

图 2-8 data 元素的属性

具体信息如下：

data 元素用于把数据选项加到 intent-filter 中。数据规范可以是数据类型或数据位置标识，也可以是数据类型和数据位置标识。

<meta-data>子元素

如图 2-9 是<meta-data>子元素的属性：

```
<meta-data android:name="string"
           android:resource="resource specification"
           android:value="string"/>
```

图 2-9<meta-data>子元素的属性

具体信息如下：

<meta-data>元素用于提供组件额外的数据，可以自定义名称和值。一个组件可以包含多个<meta-data>元素。

（1）android:name：该属性是数据项的名称。

（2）android:resource：该属性是一个被引用的资源，这个项的值是该资源的 id。

（3）android:value：该属性可以给这一项指定的值，可以指定不同数据类型。

2.2.4 <activity-alias>元素

如图 2-10 是<activity-alias>元素的属性：

```
<activity-alias android:enabled=["true" | "false"]
                android:exported=["true" | "false"]
                android:icon="drawable resource"
                android:label="string resource"
                android:name="string"
                android:permission="string"
                android:targetActivity="string">
    <intent-filter/>
    <meta-data/>
</activity-alias>
```

图 2-10<activity-alias>元素的属性

具体信息如下：

<activity-alias>元素可以为activity创建一个别名。targetActivity属性的值是目标活动，label属性的值是别名。

2.2.5 <service>元素

如图2-11是<service>元素的属性：

```
<service android:enabled=["true" | "false"]
        android:exported[="true" | "false"]
        android:icon="drawable resource"
        android:label="string resource"
        android:name="string"
        android:permission="string"
        android:process="string">
</service>
```

图2-11<service>元素的属性

该元素用于声明一个服务，服务被用于后台操作和提供通信接口，没有界面属性。<service>元素包含的属性有是否能被实例化，是否能与其他应用程序交互，服务的图标，服务的标签，服务的名字，启动或绑定服务所需的权限，服务所运行的进程。

2.2.6 <receiver>元素

该元素用于声明一个广播接收器，广播接收器能够接收来自系统或其他应用程序所发送的广播消息。广播的注册方式有两种，第一种是在AndroidManifest文件里静态注册，第二种是在代码里动态注册。<receiver>元素包含的属性有是否能被实例化，是否能与其他应用程序交互，服务的图标，服务的标签，服务的名字，启动或绑定服务所需的权限，服务所运行的进程。

2.2.7 <provider>元素

如图2-12是<provider>元素的属性：

```
<provider android:authorities="list"
        android:enabled=["true" | "false"]
        android:exported=["true" | "false"]
        android:grantUriPermissions=["true" | "false"]
        android:icon="drawable resource"
        android:initOrder="integer"
        android:label="string resource"
        android:multiprocess=["true" | "false"]
        android:name="string"
        android:permission="string"
        android:process="string"
        android:readPermission="string"
        android:syncable=["true" | "false"]
        android:writePermission="string">
        <grant-uri-permission/>
        <meta-data/>
</provider>
```

图 2-12<provider>元素的属性

具体信息如下：

该元素用于声明一个内容提供者，内容提供者必须在 AndroidManifest 里注册。ContentProvider 是 Android 中用来封装数据和数据访问接口的组件，因此它是一个被动组件，即只能被 Activity、Service 和 BroadcastReceiver 组件所访问和调用，它自己不能主动发起访问。

（1）android:authorities：该属性用于标识这个 ContentProvider 的数据 URI 的授权列表，没有默认值但必须至少指定一个授权，如果有多个授权，要用分号来分离每个授权。

（2）android:grantUriPermission：这个属性可以指定对 ContentProvider 没有访问权限的访问者是否能够被授予访问权限，如果为 ture，则可以临时授予对 ContentProvider 的所有数据的访问权限，如果未 false，则只能临时授予对<grant-uri-permission>元素中的数据的访问权限。

（3）android:initOrder：该属性可以设置同一进程中 ContentProvider 被实例化的顺序。

（4）android:readPermission：该属性用于设置访问者访问内容提供者时所需要的权限。

（5）android:writePermission：该属性用于设置访问者修改内容提供者时所需要的权限。

2.2.8 <uses-library>元素

如图 2-13 是<uses-library>元素的属性：

```
<uses-library android:name="string"
              android:required=["true" | "false"] />
```

图 2-13<uses-library>元素的属性

<uses-library>元素用于指定该应用程序链接的用户库。

android:required：该属性用于设置应用程序是否需要 android:name 属性所设置的类库。如果值为 true，表示该应用程序没有指定的类库时程序无法运行。如果值为 false，则需要在应用程序运行时对类库的有效性进行检查。默认值是 ture。

3 应用程序的组件属性配置测试案例设计

3.1 静态配置 Broadcast Receiver 的若干属性

Broadcast 可以通过在 Manifest 文件中设置<receiver>元素来配置，这种配置方式称为“静态”配置。<receiver>元素具有 7 个属性，本课题所关心的是如下二个属性：

（1） android:exported：表示 BroadcastReceiver 构件是否接收来自所在应用程序之外的广播消息。exported 具有缺省值。当<receiver>元素中没有任何<intent-filter>元素时，其缺省值为 false，即只能接收来自本应用程序（或具有相同 userId 的应用程序）中其它构件发送的广播，而且这些广播 intent 中，必须显式指定广播接收者的类名；如果有任何一个<intent-filter>元素存在，则 exported 的缺省值为 true。

（2） android:permission：该属性的设置具有唯一性，即如果设置，只能设置一次。该权限是施加给广播的发送者的。

3.1.1 android:exported 的缺省值

设计本实验的目的就是明确 android:exported 属性的缺省值。android:exported 属性的值为布尔类型，根据开发实践中的常见情况，将可能的取值分为两类：

（1）android:exported 未设置，而且<intent-filter>未设置时

（2）android:exported 未设置，而且<intent-filter>设置时

针对这两种情况，设计了两个测试案例，分别观察这样的设置对应用程序的影响。由于这些测试案例的设计比较直观，这里直接给出测试结果，如表 3-1 所示：

表 3-1 android:exported 的缺省值测试案例与测试结果

配置项组合	测试结果	说明
android:exported 未设置，& <intent-filter>未设置时	只能接收来自本应用其它构件的显式 intent，或来自具有相同 UserID 的应用程序的显式 intent 广播	android:exported 属性的缺省值为 false
android:exported 未设置，& <intent-filter>设置时	可以接收来自其它应用程序的隐式 intent 广播通信	android:exported 属性的缺省值为 true

从该组测试用例中，可以总结如下规则：

规则 1：设置<intent-filter>属性相当于设置 android:exported=true

3.1.2 android:exported 已设置的情况

设计本实验的目的就是明确 android:exported 属性和<inten-filter>属性的优先级。将实

践中可能出现的情况分为四类：

（1） 当 android:exported=false，而且<intent-filter>设置时，由 3.1.1 的结果可知，属性<intent-filter>设置时相当于 android:exported=true，这时产生了矛盾，测试这两个设置的优先级。

（2） 当 android:exported=false，而且<intent-filter>未设置时。

（3） 当 android:exported=true，而且<intent-filter>未设置时，由 3.1.1 的结果可知，属性<intent-filter>未设置时相当于 android:exported=false，这时产生了矛盾，测试这两个设置的优先级。

（4） 当 android:exported=true，而且<intent-filter>设置时。

针对这四种情况，设计了四个测试案例，分别观察这样的设置对应用程序的影响。由于这些测试案例的设计比较直观，这里直接给出测试结果，如表 3-2 所示：

表 3-2android:exported 已设置的测试案例与测试结果

配置项组合	测试结果	说明
android:exported=false & <intent-filter>设置时	该 BroadcastReceiver 不能接收来自其它应用程序的隐式 intent 通信	这两个设置中 ndroid:exported=false 的优先级更高
android:exported=false & <intent-filter>未设置时	该 BroadcastReceiver 不能接收来自其它应用程序的隐式 intent 通信	验证了 3.1.1 的结果
android:exported=true & <intent-filter>设置时	该 BroadcastReceiver 可以接收来自其它应用程序的隐式 intent 通信	验证了 3.1.1 的结果
android:exported=true & <intent-filter>未设置时	该 BroadcastReceiver 不能接收来自其它应用程序的隐式 intent 通信	两个设置中<intent-filter>的优先级更高

从该组测试用例中，可以总结如下规则：

规则 2： 对于<intent-filter>属性和 android:exported 属性一旦<intent-filter>没有设置或者 android:exported 设置为 false，该 BroadcastReceiver 不能接收来自其它应用程序的隐式 intent 通信。这两个属性之间有一个与的关系，只有当两个属性全部通过，才可以接受来自其它应用程序的隐式 intent 通信。

3.1.3 android:permission

根据 Android 官方文档，android:permission 属性的设置具有唯一性，即如果设置，

只能设置一次。设计本实验的目的是明确能否设置多个 permission，即在<receiver>元素下设置多个 permission。根据猜测，列出两种可能方案：

（1） 在<receiver>元素下设置多个 permission，多个 permssion 用“;”隔开。

（2） 能否以“或”的形式设置多个 permission 属性，即 android:permission=perm1 | perm2。

以下是几种可能出现的结果：

- 语法错，即在编译打包时报错
- Divergence，即没有语法错，但是运行时报错，如程序崩溃，弹出错误提示窗口等
- SKIP，即没有语法错，也没有运行时报错，就像没有作任何设置一样，即设置没有起到预期效果
- SUCCESS，即成功完成预期行为

针对这两种情况，设计了两个测试案例，分别观察这样的设置对应用程序的影响，如表 3-3 所示：

表 3-3 针对 android:permission 的测试案例与测试结果

配置项	测试结果	说明
permission="perm1";"perm2"	元素类型 "receiver" 必须后跟属性规范 ">" 或 "/>"，属于语法错误，即在编译打包时报错。	只能设置一个 permission
permission="perm1"|"perm2"	元素类型 "receiver" 必须后跟属性规范 ">" 或 "/>"，属于语法错误，即在编译打包时报错。	只能设置一个 permission

从该组测试用例中，可以总结如下规则：

规则 3：android:permission 属性的设置具有唯一性，即如果设置，只能设置一次，只能设置一个 permission。

3.2 动态配置 BroadcastReceiver 的若干属性

3.2.1 动态注册广播组件的生命周期

设计本实验的目的是明确动态注册广播组件时的生命周期，以下是开发实践中可能出现的两种情况：

（1）当一个 BroadcastReceiver 组件没有静态注册，且 filter!=null 时，一个 Activity 组件调用 registerReceiver(receiver, filter)方法之后，该 BroadcastReceiver 组件能够预期收到相应的广播 intent。如果之后再调用 unregisterReceiver(BroadcastReceiver receiver)之后，观察能否撤销该广播组件，撤销后能否再接收以前的广播 intent。

相应的代码如图 3-1：

```
receiver = new MyBroadCast1();
IntentFilter filter = new IntentFilter();
filter.addAction("com.zrg");
registerReceiver(receiver,filter);
unregisterReceiver(receiver);
```

图 3-1 广播注册后又撤销

动态注册广播组件，调用 registerReceiver(receiver,filter)方法注册广播后，再调用 unregisterReceiver (receiver)方法撤销该广播组件，撤销后不能再接受以前的广播 intent。

（2）当一个 BroadcastReceiver 组件没有静态注册，且 filter!=null 时，一个非 MainActivity 组件调用 registerReceiver(BroadcastReceiver receiver,IntentFilter filter)注册一个广播组件，然后该 Activity 退出，那么观察它所注册的广播组件是否也跟着退出，即应用程序不能收到发送的广播 intent。

该实验在 MainActivity 对应的布局文件中设置了两个按钮，通过一个按钮跳转到 SecondActivity，在 SecondActivity 中注册一个广播，在 SecondActivity 的布局文件中设置一个按钮，跳转回 MainActivity，通过 MainActivity 的第二个按钮发送广播。

activity_main.xml 添加的代码如图 3-2：

```
<Button
    android:id="@+id/button_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="跳转到 2" />
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="send"
    android:text="发送广播" />
```

图 3-2 相应的 activity_main.xml 添加的代码

MainActivity 中添加两个按钮的点击事件如图 3-3：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent =new Intent(MainActivity.this,SecondActivity.class);
            startActivity(intent);
        }
    });
}
public void send(View hhView){
    Intent intent = new Intent();
    intent.setAction("com.zrg");
    sendBroadcast(intent);
}
```

图 3-3 两个按钮的点击事件

activity_second.xml 添加的代码如图 3-4：

```
<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="跳转到 1" />
```

图 3-4activity_second.xml 的点击事件

SecondActivity 中添加广播的注册事件和跳转到 MainActivity 的按钮事件如图 3-5:

```
receiver = new MyBroadCast();//实例化
IntentFilter filter = new IntentFilter();
filter.addAction("cn.zrg");
registerReceiver(receiver,filter);//注册广播
Button button3=(Button) findViewById(R.id.button3);
button3.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent =new Intent(SecondActivity.this,MainActivity.class);
        startActivity(intent);
    }
});
```

图 3-5 注册广播和跳转

从该组测试用例中，可以总结如下规则：

规则 4：注册广播的 SecondActivity 退出后，它所注册的广播组件也跟着退出，即应用程序不能接收到发送的广播。

3.2.2 多次动态注册的效果

设计本实验的目的是明确几种多次注册的效果是注册了两个 receiver 对象，还是注册了一个 receiver 对象。将实践中可能出现的情况分为三类：

（1） 一个 BroadcastReceiver 组件没有静态注册，调用两次 registerReceiver(receiver, filter)，注册同一个 receiver 对象和同一个 filter。

（2） 一个 BroadcastReceiver 组件没有静态注册，调用两次 registerReceiver(receiver, filter)，注册同一个 receiver 对象和不同的 filter。

（3） 一个 BroadcastReceiver 组件没有静态注册，调用两次 registerReceiver(receiver, filter)，注册不同 receiver 对象和相同的 filter

针对这三种情况，设计了三个测试案例，分别观察这样的设置对应用程序的影响。由于这些测试案例的设计比较直观，这里直接给出测试结果，如表 3-4 所示：

表 3-4 多次动态注册的测试案例与效果

条件	结果	说明
注册同一个 receiver 对象和同一个 filter	注册了一个 receiver	同一个 receiver 对象和同一个 filter 注册多次系统只认同一次
注册同一个 receiver 对象和不同的 filter	注册了一个 receiver	一个 receiver 对象可以相应多个 filter

注册不同 receiver 对象和相同的 filter	注册了两个 receiver	不同的 receiver 对象可以相应同一个 filter

从该组测试用例中，可以总结如下规则：

规则 5：同一个 receiver 对象和同一个 filter 注册多次系统只认同一次。
规则 6：一个 receiver 对象可以相应多个 filter。
规则 7：不同的 receiver 对象可以相应同一个 filter。

3.2.3 静态注册和动态注册是否矛盾

本实验的目的是确定静态注册和动态注册是否矛盾。在 AndroidManifest.xml 文件里静态注册一个 BroadcastReceiver 组件，且<intent-filter>也已设置，这时调用 registerReceiver(BroadcastReceiver receiver, IntentFilter filter)注册同样的 BroadcastReceiver 和相同的 intentFilter。

AndroidManifest.xml 文件静态注册的代码如图 3-6：

```
<receiver android:name=".MyBroadCast">
    <intent-filter>
        <action android:name="DBR4"/>
    </intent-filter>
</receiver>
```

图 3-6 静态注册

动态注册的代码如图 3-7：

```
receiver = new MyBroadCast();
IntentFilter filter = new IntentFilter();
filter.addAction("DBR4");
registerReceiver(receiver,filter);
```

图 3-7 动态注册

结果显示运行不出错，用一个 receiver 覆盖另一个 receiver。从该组测试用例中，可以总结如下规则：

规则 8：静态注册和动态注册不矛盾，注册相同的内容，系统只认定一个。

3.2.4 递归注册

本实验是为了明确递归注册是否可行，即通过在注册的 BroadcastReceiver 的 onReceive()方法调用 registerReceiver()。实践中可能出现的情况有两种：

（1） 能否让一个静态注册的 BroadcastReceiver 动态注册另一个 BroadcastReceiver。

AndroidManifest.xml 文件静态注册的广播 1 如图 3-8：

```
<receiver
    android:name=".MyBroadCast"
    android:enabled="true"
    android:exported="true">
</receiver>
```

图 3-8 递归静态注册 1

MyBroadCast 的 onReceive()方法中动态注册广播 2 如图 3-9：

```
receiver2 = new MyBroadCast2();
IntentFilter filter2 = new IntentFilter();
filter2.addAction("DBR5_1_2");
context.registerReceiver(receiver2,filter2);
```

图 3-9 递归动态注册 1

结果显示一个静态注册的 BroadcastReceiver 不能动态注册另一个 BroadcastReceiver，即不可以递归调用。

（2） 能否让一个动态注册的 broadcastReceiver 动态注册另一个 BroadcastReceiver。

MainActivity 中动态注册广播 1 如图 3-10：

```
receiver = new MyBroadCast1();
IntentFilter filter1 = new IntentFilter();
filter1.addAction("DBR5_2");
registerReceiver(receiver,filter1);
```

图 3-10 递归动态注册 2

MyBroadCast1 的 onReceive()中动态注册广播 2 如图 3-11：

```
receiver2 = new MyBroadCast2();
IntentFilter filter2 = new IntentFilter();
filter2.addAction("DBR5_2_2");
context.registerReceiver(receiver2,filter2);
```

图 3-11 递归动态注册 2

结果显示一个动态注册的 BroadcastReceiver 可以动态注册另一个 BroadcastReceiver，即可以递归注册。

从该组测试用例中，可以总结如下规则：

规则 9：一个静态注册的 BroadcastReceiver 不能动态注册另一个 BroadcastReceiver，即不可以递归调用。

规则 10：一个动态注册的 BroadcastReceiver 可以动态注册另一个 BroadcastReceiver，即可以递归注册。

3.3 ContentProvider 的若干属性配置。

3.3.1 对 readPermission 和 writePermission 的配置

设计本实验的目的是明确同一 APP 中的其它组件是否可以不受任何限制的对该 APP 中定义的 ContentProvider 的 URI 进行读写访问。针对以上问题设计出两种测试案例：

测试条件：<grant-URI-Permission>未定义，readPermission 和 writePermission 指定特定的权限：

（1）当 android:grantUriPermissions="true"时，同一 APP 内的其它组件能否访问 URI

（2）当 android:grantUriPermissions="false"时，同一 APP 内的其它组件能否访问 URI

3.3.2 grantUriPermission 属性

设计本实验的目的是明确子元素<grant-URI-Permission>与属性 grantUriPermission 的关系。根据实际情况分为四种可能：3

（1）<grant-URI-Permission>未定义，并且 grantUriPermission=false 时，观察是否无法向任何外部组件（指其它 APP 的组件）成功授权？如果出错，是在授权时出错，还是在外部组件访问 provider 时出错？

（2）<grant-URI-Permission>未定义，并且 grantUriPermission=true 时，观察是否无法向任何外部组件（指其它 APP 的组件）成功授权？

（3）<grant-URI-Permission>定义，并且 grantUriPermission=false 时，如果授予的 URI 不在<grant-URI-Permission>定义的 URI 范围内，观察是否无法向任何外部组件（指其它 APP 的组件）成功授权？如果出错，是在授权时出错，还是在外部组件访问 provider 时出错？

如果授予的 URI 不在<grant-URI-Permission>定义的 URI 范围内，观察是否无法向任何外部组件（指其它 APP 的组件）成功授权？

（4）<grant-URI-Permission>定义，并且 grantUriPermission=true 时，授权的范围是否与<grant-URI-Permission>定义的 URI 范围有关？

3.3.3 exported 属性

设计本实验的目的是为了明确 exported 属性的准确含义和变化。根据实际情况可以分为三类:

(1) exported 属性的默认值，即不设置这个属性;

(2) 设置 android:exported="true";

(3) 设置 android:exported="false"。

3.3.4 三个权限属性之间的关系

设计本实验的目的是为了明确 readPermission，writePermission 和 permission 属性之间的关系。根据实际情况可以分为三类:

(1) 仅仅设置 permission 属性；

(2) permission 属性和 readPermission 属性同时设置；

(3) permission 属性和 writePermission 属性同时设置。

3.3.5 multiprocess 属性

设计本实验的目的是为了明确 multiprocess 属性的准确含义和变化。根据实际情况可以分为三类:

(1) multiprocess 属性的默认值，即不设置这个属性;

(2) 设置 multiprocess="true";

(3) 设置 multiprocess="false"。

4 检测工具的设计与实现

4.1 可行性分析

4.1.1 经济可行性

本课题的用到的资源包括一个笔记本电脑，从图书馆借的一些书籍，网上可以搜索到相关的文献资源。

本次实验环境配置如表 4-1 所示:

表 4-1 实验环境配置清单

操作系统	Windows10
处理器	Intel® Core™ i5-6300HQ CPU @ 2.30GHz
内存	4G
实验工具	ApkToolBox(apktool、dex2jar、jadx)
IDE 工具	AndroidStudio+Android SDK

4.1.2 技术可行性

熟练 Android 开发、熟悉 Android 配置、对数理逻辑有一定掌握，也可以跟着网上的资料进行学习。

4.1.3 操作可行性

需要下载一些安装包，通过反编译工具包反编译，得到反编译文件，从中得到 manifest 文件，用检测工具遍历 manifest 文件并与总结的规则做比较，输入结果就可以。

4.2 需求分析

功能需求：对于反编译出的 Manifest 文件，设计和实现一个工具，要求工具可以将 Manifest 文件中的配置项和总结的规则对比判断 apk 的安全漏洞。

性能需求：1min 内可以完成一个过程。

环境需求：Windows10，内存 4G，ApkToolBox，AndroidStudio，AndroidSDK。

可靠性需求：若无硬件、系统故障或计算机的超负荷运行可以正常运行。

安全保密需求：无保密需求。

用户界面需求：简洁明了。

预先估计以后系统可能达到的目标：输入一个 apk 文件，可以与自己的总结的规则比较之后，输出结果，判断 apk 所在软件是否存在安全问题。

4.3 检测工具的设计

如图 4-1 是工具的结构图：

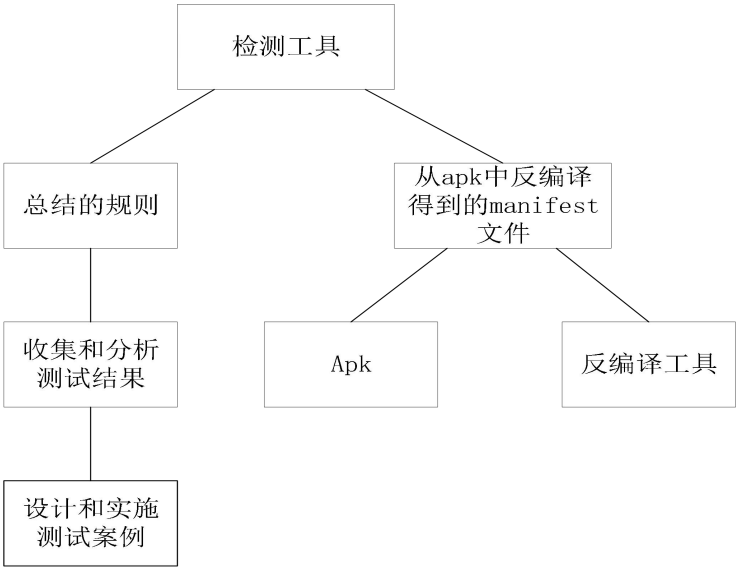

图 4-1 结构图

如图 4-2 是工具的流程图：

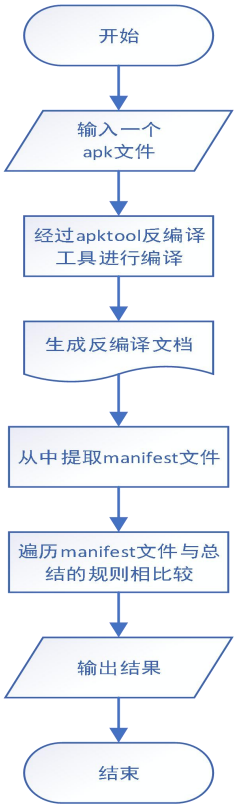

图 4-2 工具的流程图

如图 4-3 是检测工具的界面：

图 4-3 测试工具

首先输入 apk 路径，然后通过反编译工具反编译 apk 文件，在从反编译文件中提取 manifest 文件，通过遍历 manifest 文件并且与总结的规则做对比，判断 apk 所对应的应用程序是否存在安全问题，输出对比结果。

5 总结

本论文以 Android 配置的组合测试和基于配置的安全检测工具的开发为目标，重点研究了基于 BroadcastReceiver 和 ContentProvider 的相关配置项组合测试。通过了解和熟悉 Android 的组件配置所代表的含义,针对 BroadcastReceiver 和 ContentProvider 设计了若干个测试案例，并进行了测试，收集和分析测试结果，总结出一组配置规则，开发了一个工具，能够检测一个程序是否与这些规则匹配，如果不匹配，则程序可能存在一些逻辑上或开发上的矛盾、安全上的一些隐患。本课题研究内容如下：

（1） 查阅并学习 Android 组件和配置项的含义以及相关文献，对本课题的背景和意义、国内外研究现状进行了介绍。

（2） 在熟悉 Android 组件和配置项的情况下，设计和实施测试案例，收集和分析测试结果，提炼配置规则。

（3） 根据得到的配置规则，设计并实现一个简单的检测工具，达到检测 Android 应用安全的作用。

致　谢

时间匆匆而过，大学生活即将结束，毕业设计虽然只有几个月的时间，但是在这几个月里我学到了很多东西。在这里，我真诚的感谢所有指导、帮助和关心我的老师、朋友和同学。

首先我要感谢我的指导老师刘晓建副教授，刘老师对我的毕业设计和论文的书写给与了很多悉心指导。在课题的研究和论文的写作过程中，刘老师给了我很多耐心的指导和启发。刘老师渊博的学识、严谨的治学态度、敬业的精神和平易近人的态度，给我留下了深刻的印象，在刘老师的教导下，我在学习和生活方面都得到了很多的进步！我不仅在专业知识方面学到了很多，还学到了很多为人出事的道理，将不断激励着我奋发学习，努力进步。刘老师的敦敦教诲将令我终身难忘。在此，请允许我表示我深深的敬意和由衷的感谢。

最后，感谢毕业设计和毕业论文完成的这个过程，感谢陕西信息网络中心的老师的指导，感谢所有这期间给与我帮助和关心的老师、同学和朋友。

参考文献

[1] 许庆富,谈文蓉,王彩霞.基于逆向工程的 Android 应用漏洞检测技术研究[J].西南民族大学学报(自然科学版),2018,44(05):512-520

[2] 李志杰. Android 应用拒绝服务漏洞检测技术研究与实现[D].电子科技大学,2018

[3] 付胧玉. 基于静态污点分析的 Android 应用漏洞检测技术研究[D].西安电子科技大学,2017

[4] 吴丹,刘嘉勇,贾鹏,肖顺陶.基于特征匹配的 Android App 漏洞检测系统设计与实现[J].网络安全技术与应用,2017(01):93-95

[5] 宋丽珠,林柏钢,倪一涛,李应,曾哲凌.Android 软件漏洞检测方法与技术研究[J].网络空间安全,2016,7(Z1):54-62

[6] 张嘉元.一种基于匹配的 Android 系统漏洞检测方法[J].电信科学,2016,32(05):132-137

[7] S.Benli A. Habash A. Herrmann T. Loftis and D. Simmonds "A comparative evaluation of unit testing techniques on a mobile platform " in Proceedings of the 9th International Conference on Information Technology-New Generations ser. ITNG '12. IEEE Computer Society 2012 pp. 263-268

[8] William Enck,Machigar Ongtang,Patrick McDaniel.Understanding Android Security[J],2009,7(01):50-57

[9] 郭霖著.第一行代码——Android [M].2 版.北京:人民邮电出版社,2016

[10] 李晓洲. Android 应用程序组件漏洞测试方法研究[D].太原理工大学,2015

[11] 董国伟,王眉林,邵帅,朱龙华.基于特征匹配的 Android 应用漏洞分析框架[J].清华大学学报(自然科学版),2016,56(05):461-467

[12] 陈璐,马媛媛,石聪聪,李尼格,李伟伟.Android 应用安全缺陷的静态分析技术研究[J].计算机工程与应用,2018,54(04):117-121

[13] 传智播客高教产品研发部. Android 移动应用基础教程[M]. 北京:中国铁道出版社,2015

[14] 靖二霞,应凌云,路晔绵,苏璞睿.基于 Dalvik 寄存器污点分析的 Android 漏洞检测方法[J].计算机应用研究,2018,35(03):916-921

[15]]Krzysztof Opasiak,Wojciech Mazurczyk. (In)Secure Android Debugging: Security analysis and lessons learned[J]. Computers & Security,2019,82

[16] Christos Lyvas,Costas Lambrinoudakis,Dimitris Geneiatakis. Dypermin: Dynamic permission mining framework for android platform[J]. Computers & Security,2018,77