分 类 号：_____ 密级：_____

学　　号：19308207009

西安科技大学
XI'AN UNIVERSITY OF SCIENCE AND TECHNOLOGY

硕士学位论文

Thesis for Master's Degree

安卓 App 配置安全性检查方法研究

申请人姓名：**

指导教师：***(校内)　***(校外)

类　　别：非全日制专业学位硕士

工程领域：计算机技术

研究方向：软件安全

2022 年　6 月

西安科技大学

学 位 论 文 独 创 性 说 明

本人郑重声明：所呈交的学位论文是我个人在导师指导下进行的研究工作及其取得研究成果。尽我所知，除了文中加以标注和致谢的地方外，论文中不包含其他人或集体已经公开发表或撰写过的研究成果，也不包含为获得西安科技大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

学位论文作者签名：　　日期：

学 位 论 文 知 识 产 权 声 明 书

本人完全了解学校有关保护知识产权的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属于西安科技大学。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文被查阅和借阅。学校可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律注明作者单位为西安科技大学。

保密论文待解密后适用本声明。

学位论文作者签名：　　指导教师签名：

年　　月　　日

论文题目：安卓 App 配置安全性检查方法研究

学科名称：计算机技术

硕 士 生：** （签名）_____

指导老师：***(校内) （签名）_____

***(校外) （签名）_____

摘 要

Android 系统提供了诸如文件访问控制、安全沙箱（Sandbox）、权限机制、应用程序签名机制等措施来保护系统与应用程序的安全，然而 Android 系统仍然暴露出诸如权限机制漏洞、隐私信息泄露等严重的安全问题。造成这些安全问题的原因之一是 Android 应用程序的组件配置不合理。尽管在 Android 官方文档中对组件配置元素已经进行了描述，但是这些自然语言描述存在不确定性、二义性，会造成开发者一些理解性的偏差，可能致使开发者对应用程序配置不当，进而产生安全漏洞。因此，对 Android App 组件配置元素的语义以及各元素之间关系的认定和发掘，对于保障应用程序安全性具有重要意义。主要研究内容如下：

（1）针对 Android 官方文档中部分组件配置描述语义模糊的问题，用模糊测试和组合测试方法构造测试用例，通过对用例运行结果的分析，明确这些配置项的含义，在此基础上进一步采用一阶谓词逻辑描述这些配置项的语义，形成若干形式化配置规则，以供设计及开发人员使用，进而减少设计漏洞。

（2）为了展示由于配置方面的漏洞所引发的 App 安全问题，以三种典型的任务劫持攻击为例，研究了配置方面的脆弱性，实现了五组实例，其中包括网络钓鱼攻击（2 组实例），欺骗攻击（1 组实例），勒索 App（2 组实例），再现了由于配置引起的几种劫持攻击过程，例证了配置对漏洞发现所产生的关键作用，并给出了缓解此类攻击的安全指南。

（3）基于上述若干配置规则以及劫持攻击相关的配置漏洞，提出了一种 Android App 配置安全性检查方法，并设计和开发了一个 Android App 配置安全检测工具，其中包括逆向分析模块、组件信息分析模块、日志输出模块三个模块。使用 CICMalDroid 2020 公开数据集和各大应用厂商中下载的应用程序进行了实验，证明了该工具的有效性。进一步将该工具与现有成熟工具 MobSF 中的配置安全性检测进行了对比，结果表明该工具能够额外检测 launchMode、allowTaskReparenting、taskAffinity 等配置项可能造成的安全漏洞，形成了对 MobSF 的有益扩展和补充。

关键词：安卓应用程序；组件配置；模糊测试；组合测试；任务劫持

研究类型：应用研究

Subject : Research on the security inspection method of Android App configuration

Specialty : Computer Technology

**Name : ** (Signature)_____

Instructor : * (Signature)_____

*** (Signature)_____

ABSTRACT

The Android system provides measures such as file access control, security sandbox (Sandbox), permission mechanism, and application signature mechanism to protect the security of the system and applications. However, the Android system still exposes serious problems such as permission mechanism loopholes and privacy information leakage. security issues. One of the reasons for these security problems is that the components of Android applications are not properly configured. Although the component configuration elements have been described in the official Android documentation, these natural language descriptions have uncertainty and ambiguity, which will cause some deviations in developers' understanding, which may cause developers to improperly configure the application, and then Create security holes. Therefore, the identification and exploration of the semantics of Android App component configuration elements and the relationship between each element are of great significance for ensuring application security. The main research contents are as follows:

(1) Aiming at the problem of vague semantics in some component configuration descriptions in Android official documents, construct test cases with fuzz testing and combined testing methods, and clarify the meaning of these configuration items by analyzing the results of the use cases. The order predicate logic describes the semantics of these configuration items and forms of several formal configuration rules for designers and developers to use, thereby reducing design loopholes.

(2) To show the App security problems caused by the vulnerabilities in the configuration, taking three typical task hijacking attacks as examples, the vulnerabilities in the configuration are studied, and five groups of instances are implemented, including phishing attacks (2 groups). Examples), spoofing attacks (1 group of examples), ransomware App (2 groups of examples), reproduced several hijacking attack processes caused by configuration, exemplified the key role of configuration on vulnerability discovery, and gave mitigations for such Security Guidelines for Attacks.

(3) Based on the above configuration rules and configuration vulnerabilities related to hijacking attacks, a security check method for Android App configuration is proposed, and an Android App configuration security detection tool is designed and developed, including a reverse analysis module, component information analysis Module, log output module three modules. Experiments are carried out using the CICMalDroid 2020 public dataset and apps downloaded from major app vendors to demonstrate the effectiveness of the tool. The tool is further compared with the configuration security detection in the existing mature tool MobSF. The results show that the tool can additionally detect the possible security vulnerabilities caused by configuration items such as launchMode, allowTaskReparenting, taskAffinity, etc., forming a useful extension and supplement to MobSF.

Keywords: Android Application; Component Configuration; Fuzzing; Combination Test; Task Hijacking

Thesis: Application Research

目 录

1 绪论

1.1 选题背景与意义

相较于其他操作系统而言，Android 系统的开源性、易用性、开放性、丰富的应用生态等优点，使得 Android 系统迅速的被广大消费者与开发者所接受，从而极大地促进了 Android 系统的发展。2021 年中国手机操作系统行业研究报告[1]指出，国内主要使用 Android 系统的手机终端厂商分别为华为、OPPO、vivo 和小米。以华为为例，预计在 2017 年至 2025 年期间，华为 Android 智能手机出货量将从 0.9 亿台增长至 1.2 亿台，显示了 Android 系统在国内的巨大使用量。

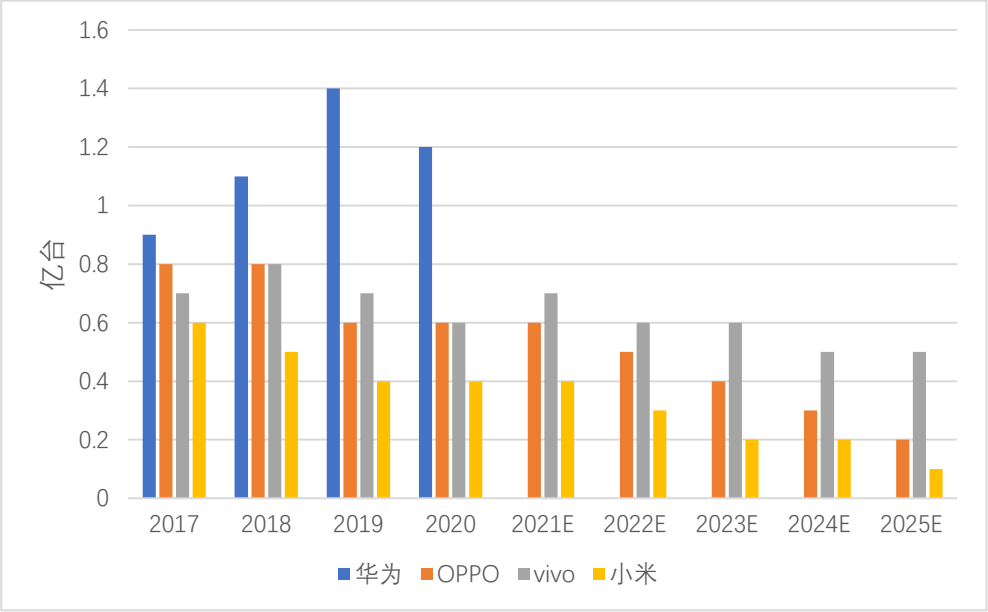


图 1.1 2017-2025 年预计中国四大 Android 系统手机出货量统计图

然而，由于大量的开发者加入到 Android App 的开发行列，部分开发人员对 Android 系统应用程序的安全机制认识不够深入，导致 App 中可能存在许多安全隐患。比如，在应用程序层面，应用程序组件包含的主要安全问题有：（1）Activity、Service、Receiver 等这些组件之间通过 Intent 显式或隐式调用时，恶意攻击者通过拦截或监听 Intent，对应用程序发起 DoS 攻击、钓鱼攻击或欺骗攻击以达到非法读取用户数据的目的；（2）Provider 是 Android 应用程序中提供数据存储的组件，当该组件被配置为向外公开（exported）时，存储的数据可被未授权的组件读取或写入；（3）由于组件的权限配置不当，可能导致隐私信息泄露、权限泄露[2]、提权漏洞和组件劫持[3]等。由此可见，引发以上安全问题的原因之一是 Android 应用程序的组件配置不合理。

尽管 Android 官方文档中对组件配置元素已经进行了描述，但是一些描述没有给出

准确的语义定义，一些描述没有给出相关配置元素之间准确的关系，因此这些自然语言描述存在不确定性、二义性，会造成开发者一些理解性的偏差，可能致使开发者对 App 配置不当，产生安全漏洞。下面通过两个例子来说明，Android 官方文档用自然语言描述配置的含义时存在二义性：

（1）每一个组件都可以通过<permission>标签配置 enforced permission，该权限说明了此 App 施加给与它通信时其它 App 应该具有的权限，但 Android 官方文档中并没有对每个组件配置该类权限的数量进行明确的约束。测试实验表明只能配置一个 enforced permission。

（2）根据 Android 官方文档，可以在 AndroidManifest.xml 文件中设置应用程序或组件的进程（process）属性，但是对该属性值应遵守的格式并没有给出严格说明。根据测试实验结果，可以得出 process 属性值的格式必须遵守 Java 包格式，否则安装该应用程序时会出现错误。

通过以上两个例子可以看出，对 Android App 组件配置元素的语义以及各元素之间关系的认定和发掘，对于应用程序开发和保障应用程序安全性具有十分重要的意义。

1.2 国内外研究现状

1.2.1 Android App 配置安全研究现状

Android App 配置安全是指与应用程序静态配置相关或者由应用程序的配置项漏洞导致的一些安全性问题。由于与安全性相关的配置中主要涉及权限配置以及组件间通信的相关配置，因此下面对这两类的相关工作进行分析。

（1）Android App 权限配置的相关工作

权限机制是 Android 系统中最重要的安全措施之一，Android App 通常存在过度索权的安全问题，即应用程序会请求其不必要的权限。Xiao 等人[4]提出了一种评估过渡索权风险的方法 MPDroid。该方法首先使用协同过滤算法来识别该应用程序的初始最小权限集，然后通过静态分析，识别出应用程序真正需要的最终最小权限集，最后通过检查应用程序的额外权限，即应用程序请求的不必要权限，来评估过渡索权风险。

Android 权限是静态检测中最为有效的特征，但其种类繁多，且每个权限对分类的贡献大小不一，因此，选择对分类贡献明显的权限尤为重要。李秀等人[5]提出了一种结合 Relief 算法和 Apriori 算法的特征选择方法 RApriori。该方法利用 Relief 算法对 Android 权限去冗余，进行第一次特征选择，然后利用 Apriori 算法对去冗余后的权限关联挖掘，进行第二次特征选择。最后，利用随机森林算法对经特征选择后的权限分类建模。实验表明，RApriori 方法的恶意应用检测率达到 90%。

刘倩等人[6]提出一种基于应用分类和敏感权限挖掘的恶意应用检测方法。首先对应用程序样本集进行频繁模式挖掘得到每一类应用程序的敏感权限集，并根据敏感权限的使用情况，计算出该类应用程序的敏感阈值。当有应用程序安装时，利用分类模型给应用程序进行正确分类，同时统计出其中敏感权限的使用情况，计算出该应用程序的敏感值。最后与该类应用程序的敏感阈值进行比较，判断此应用程序是否为恶意应用程序。Enck 等人[7]提出一种基于权限的恶意应用检测方法 Kirin，通过定义一组匹配恶意软件的权限使用规则来识别样本恶意性，虽然在一定程度上可以识别恶意软件，由于该方法从配置文件 AndroidManifest.xml 中提取应用权限，所以可以通过修改配置文件中的权限声明绕过检测。

针对 Android 权限机制存在的问题和传统应用风险等级评估方法的不足，卜同同等人[8]提出了一种基于权限的 Android 应用程序风险评估方法。首先提取 App 声明的各类权限，然后从具有恶意倾向的组合权限、“溢权”问题和自定义权限三个方面对 App 进行量化风险评估，最后采用层次分析法（AHP）计算上述三个方面的权重，评估应用程序的风险值。

为了解决 Android 系统粗颗粒权限机制引起的软件滥用隐私权限的问题，需要对 Android App 权限进行细粒度的控制。金俊杰[9]提出了一种 HSFR（Hook-based System Functional Response）模型，该模型结合原生权限机制，控制了 App 所申请权限请求，并结合 Hook 方式拦截第三方软件和系统的交互，使设备按照用户配置方式返回真实或非真实的数据信息。实验证明，HSFR 可以实现高效、灵活的 Android 系统权限细粒度管理。Meng 等人[10]实现了一个隐私保护系统 AppScalpel，该系统使用静态分析来提取应用程序中数据使用行为的特征，然后利用离群值检测方法准确地识别了不合理使用的敏感数据信息，最后通过代码插装技术在每个恶意的数据流路径上分别设置规则强制执行器，确保应用程序遵守隐私保护规则。实验表明，AppScalpel 能够精确地识别不合理使用的敏感数据信息，以细粒度模式有效地保护用户的隐私信息。

（2）Android App 组件间通信安全的相关工作

针对组件间通信可能引发的隐私泄露问题，Bohluli 等人[11]开发了 IIFDroid 组件间信息流控制静态分析工具，旨在检测 Android 应用程序内部的各种显式和隐式信息流所产生的信息泄漏。李智等人[12]提出了一种基于动静结合技术的检测工具 Privacy Miner，首先对 Android 设备中的敏感数据进行静态污点分析，检测 Android 的隐私保护机制能否有效地防止这些敏感信息被恶意软件读取。然后，检查恶意应用程序在不需要声明任何权限时，能否将读取的敏感数据发送出去。最后该工具可以在线对应用商城中的 App 进行分析，并将报告发送给用户。Octeau 等人[13]把组件间通信造成的隐私泄露问题转化

为一个分布式环境问题，根据问题的分析结果，设计并实现了一个 Dalvik 反编译器，并通过分析反编译的源代码，研究了应用程序中会引起隐私泄露的危险功能以及这些功能在组件间通信时是如何发生的。

针对 Android 应用程序中组件间通信的健壮性问题，Choi K 等人[14]实现了一个 Intent 模糊测试工具 Hwacha。该工具首先设计了一个 Intent 规范语言来描述 Intent 的结构，根据规范，生成器将生成各种 Intents，执行器将对这些 Intents 的通信行为进行测试。然后该工具提出了一种故障自动分类计数方法，在使用 Intent 进行测试时将识别多次出现的相同故障进行自动计数，解决了手工分类相同故障的耗时问题。实验发现了 400 多个 Intent 漏洞，证明了该工具的有效性。肖卫等人[15]设计并实现了一个基于动静结合分析技术的检测工具 Intent Checker。该工具首先通过静态数据流分析技术跟踪 App 对 Intent 对象的数据访问，识别是否存在畸形数据的验证漏洞，并提取漏洞利用的关键特征，最后对 App 进行攻击测试，用于对 Intent 漏洞的进一步确认。

针对应用程序组件间通信中组件暴露的问题，王国珍等人[16]提出了一种系统化测试开放 Activity（Exported Activity，EA）的方法。首先使用静态分析技术解析 APK 文件，提取出 EA 列表和启动它们需要的数据键值和类型。然后将相应的数据填充到预先设置好的模板中，生成测试 EA 的测试驱动 App，使用这些代理 App 作为测试驱动程序，启动被测 EA。最后对执行过程中出现的问题做了深入研究，统计了测试过程中出现 App 崩溃和显示异常两类问题的测试驱动程序数量，实验表明虽然 EA 被广泛应用于 App 中，但开发者对它的开发并不完备。

1.2.2 模糊测试现状

模糊测试具有高效、简单、易用的特点，已经被广泛应用于不同的测试场景，如应用程序测试（Application fuzzing）、网络协议测试（Protocol fuzzing）、文件格式测试（File format fuzzing）、针对网络应用的测试（Web app fuzzing）等。为了应对不同的应用场景，研究者们不断开发各种各样的模糊测试工具。在 BlackArch Linux 中已经收录了 75 种开源的模糊测试工具[17]，其中较为著名的是 AFL[18]。该工具使用一种新型的编译时工具和遗传算法来自动构造测试用例，这提高了模糊代码的功能覆盖率，已经在现实的程序中发掘了大量漏洞。

Li 等人[19]提出基于程序状态的二进制模糊测试方法 Steelix，该方法使用二进制插桩技术及轻量级的静态分析技术来获取程序的运行时状态信息，使得模糊测试工作可以覆盖到程序更深的部分，且具有更好的漏洞检测能力。同样，通过分析应用程序行为来推断其输入属性也是一种可行的改进模糊测试性能的策略。Rawat 等人[20]提出了一种基于应用程序感知的进化模糊策略方法 VUzzer，该方法基于动静结合技术分析应用程序

的控制流和数据流特性来推断应用程序的基本属性，能够覆盖更大的范围和探索更深的路径，且更快的生成模糊测试的输入。实验表明，该模糊器可以快速的发现几个现有的和新的错误，与 AFL 模糊器相比，VUzzer 产生了更好的效果。She 等人[21]提出了一个基于程序平滑技术（smoothing technique）的模糊工具 Neuzz，该工具首先使用前馈神经网络（NNs）有效的模拟了目标程序的分支行为，然后提出了一种增量学习技术，以迭代的方式细化神经网络模型，实验表明，前馈神经网络模型的梯度可以用来高效地生成程序输入，使目标程序中发现的错误量最大化，显著的提高了模糊测试过程的效率。

针对模糊测试过程中对软件的覆盖受限制的问题，研究者结合基于补丁的模糊测试方法和混合模糊测试方法两种方法去解决这个问题，且已经有了进展，但该方法在分析更深层次的代码分支和绕过实际程序中的路障检查（roadblock checks，RC）方面仍然存在不足。Rustamov 等人[22]实现了一个高效的深度模糊测试工具 DeepDiver，该工具通过结合 AFL++以及先进的符号执行方法，消除程序中的 RCs，根据消除过程，可以探索新的测试执行路径，来挖掘深层隐藏的程序漏洞，解决了现有混合模糊测试方法所显示的局限性问题。实验表明，DeepDiver 发现漏洞的平均速度比 QSYM 和 AFLFast 更快，并实现了深度代码覆盖。

1.2.3 组合测试现状

1985 年，Mandl[23]提出了成对组合测试的概念，利用变换矩阵来构造覆盖参数组合的测试用例。至今，组合测试领域一直是软件测试领域的研究热点。我国于 2020 年发布了关于组合测试的国家标准[24]（GB/T 38639-2020），为组合测试技术的应用提供了重要依据。组合测试[25]就是对 n 个参数构造测试用例，使得对于任意 t（通常$2 \leq t \leq n$较小的整数）个参数的所有可能组合至少被一个测试用例覆盖。近年来，国内外学者对组合测试技术进行了系统化的研究，得出了许多行之有效的研究方法。

最小覆盖表的生成是组合测试研究的关键问题，而启发式算法可以用来解决这个问题，此算法主要分为基于一维扩展策略和基于二维扩展策略两种算法。基于一维扩展策略（on-test-at-a-time）的启发式算法最初是由 Sherwood 等人[26]提出。包晓安等人[27]将 one-test-at-a-time 策略和自适应粒子群算法相结合，对种群的粒子进行分析记录，根据粒子的好坏调整粒子群算法的相关参数，来提升寻优能力，实现了更优的测试用例生成方法。基于二维扩展策略（in-parameter-order，IPO）的启发式算法最初是由 Lei 等人[28]提出。周进[29]提出了一种以 IPO 算法思想为基础的 IPO_I 算法。该算法在继承了 IPO 算法扩展性强优点的同时，将实际待测系统中存在的交互关系，作为 IPO 算法的初始矩阵构造、未覆盖参数的扩展及取值步骤提供的依据，使得算法获得更优的测试用例集，且提升了测试用例集生成效率。

综上，模糊测试和组合测试都已经成为了软件测试的重要手段。本文第 3 章中使用模糊测试以及组合测试方法构造测试用例，模拟各种可能的配置以及不同配置项之间的组合关系，并通过测试结果研究一部分 Android App 配置属性的准确含义。

1.2.4 任务劫持攻击现状

Android 多任务系统提供了丰富的功能来增强用户体验，促进了应用程序个性化。然而，Android 多任务处理的独特设计仍存在严重的安全风险，使得应用程序容易受到任务劫持攻击。

英国《2020 年网络安全漏洞调查》[30]显示，网络钓鱼攻击是最常见的网络攻击方法。网络钓鱼攻击属于任务劫持攻击，该类攻击可能会导致个人隐私信息泄露、公司和政府机密泄露等，给用户带来严重的损失。Felt 等人[31]研究了移动设备受到网络钓鱼攻击的原因，并对攻击的方法和路径进行了评估。Aonzo 等人[32]对 Android 的移动密码管理器（mobile password managers）和即时应用（Instant Apps）这两项功能，进行了安全评估，并展示了移动设备的主要密码管理器受到钓鱼攻击的过程。若攻击者大量使用即时应用技术，便可完全控制用户界面。同时发现，移动密码管理器也容易受到“隐藏字段”攻击，可以看出 Android 推出的这两项功能并不安全。因此该文献提出了一种新的安全设计方法来避免常见攻击。

Chen 等人[33]发现了后台应用程序在不需要任何许可的情况下，能够获取用户界面状态的漏洞，根据该漏洞设计并实现了几种新的攻击模式，其中包括劫持界面状态来窃取用户敏感输入（如登录凭证）和获取相机中用户拍摄的敏感图像（如银行应用的个人信息截图）。Fratantonio 等人[34]发现了一种 Android 应用程序漏洞，设计模拟了恶意 App 针对该漏洞的攻击方式，该类攻击首先在用户不知情的情况下窃取用户的登录凭证和安全密码，然后在后台安装一个恶意 App 并为其启用了所有权限。实验表明，这类攻击方式可以完全控制用户的输入和显示的输出，从而破坏设备界面。

1.3 本文主要工作

本文针对 Android 官方文档中对组件配置元素的描述存在不确定性、二义性的问题，展开对 Android App 组件配置元素的语义以及各元素之间关系的认定和发掘，并研究了任务劫持攻击方法，以及由于配置漏洞所产生的安全问题。主要内容如下：

（1）针对 Android 官方文档中部分组件配置描述语义模糊的问题，用模糊测试和组合测试方法构造测试用例，通过对用例运行结果的分析，明确这些配置项的含义，在此基础上进一步采用一阶谓词逻辑描述这些配置项的语义，形成若干形式化配置规则，以供设计及开发人员使用，进而减少设计漏洞。

（2）为了展示由于配置方面的漏洞所引发的 App 安全问题，以三种典型的任务劫持攻击为例，研究了配置方面的脆弱性，实现了五组实例，其中包括网络钓鱼攻击（2 组实例），欺骗攻击（1 组实例），勒索 App（2 组实例），再现了由于配置引起的几种劫持攻击过程，例证了配置对漏洞发现所产生的关键作用，给出了缓解此类攻击的安全指南。

（3）基于上述若干配置规则以及劫持攻击相关的配置漏洞，提出了一种 Android App 配置安全性检查方法，并设计和开发了一个 Android App 配置安全检测工具，其中包括逆向分析模块、组件信息分析模块、日志输出模块三个模块。使用 CICMalDroid 2020 公开数据集和各大应用厂商中下载的应用程序进行了实验，证明了该工具的有效性。进一步将该工具与 MobSF 的配置安全性检测进行了对比，结果表明该工具能够额外检测 launchMode、allowTaskReparenting、taskAffinity 等配置项可能造成的安全漏洞，形成了对 MobSF 的有益扩展和补充。

1.4 章节安排

本文主要研究安卓 App 配置安全性检查方法，共分为六个章节，具体的章节安排如下：

第 1 章：绪论。主要介绍对 Android App 组件配置研究的重要性，分析了 Android 官方文档对组件配置元素描述二义性的问题，介绍了 Android App 配置安全、模糊测试、组合测试和任务劫持攻击的研究现状，概括了全文的研究工作和章节安排。

第 2 章：相关理论与技术分析。首先介绍了 Android 系统的基本组件和 AndroidManifest.xml 文件，然后阐述了 Android 组件安全机制，包括应用程序沙箱防御机制、权限机制和 Intent 通信的安全措施，最后介绍了模糊测试和组合测试方法。

第 3 章：安卓 App 配置规则研究。针对 Android 官方文档中部分组件配置元素描述语义模糊的问题，用模糊测试和组合测试方法构造测试用例，通过对用例运行结果的分析，明确这些配置项的含义，在此基础上进一步采用一阶谓词逻辑描述这些配置项的语义，形成若干形式化配置规则。

第 4 章：任务劫持攻击方法研究。以三种典型的任务劫持攻击为例，实现了五组实例，其中包括网络钓鱼攻击（2 组实例），欺骗攻击（1 组实例）和勒索 App（2 组实例），再现了由于配置漏洞引起的几种劫持攻击过程，通过分析攻击实验，统计各大应用厂商中应用程序使用任务劫持条件的比例，给出了缓解此类攻击的安全指南。

第 5 章：安卓 App 配置安全性检查方法研究。基于上述若干配置规则以及劫持攻击相关的配置漏洞，提出了一种 Android App 配置安全性检查方法，并设计和开发了一个 Android App 配置安全检测工具。然后使用公开数据集进行了实验，验证了该工具的有效性。结果表明该工具的检测工作能够对 MobSF 进行有益的扩展和补充。

第 6 章：总结与展望。对本文的研究内容进行总结，并对未来进行展望，进一步明确后续工作目标。

2 相关理论及技术分析

2.1 Android 相关概念

2.1.1 Android 基本组件

Android 应用程序组件是 Android 应用程序的基本构建块，AndroidManifest.xml 描述了每个组件的基本信息以及交互过程。Android 系统组件关系如图 2.1 所示。

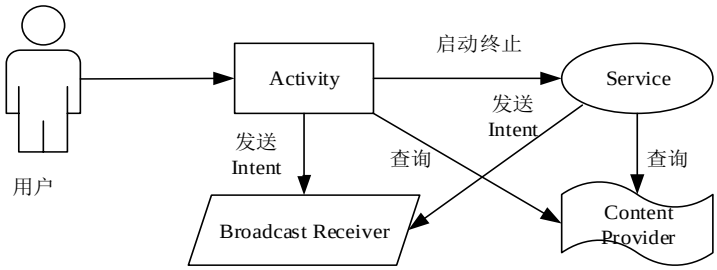


图 2.1 Android 系统组件关系

Android 应用程序有四大组件，分别是 Activity（活动）、Service（服务）、Broadcast Receiver（广播接收器）和 Content Provider（内容提供者），下面进行详细介绍。

（1）Activity：Activity 是用户操作的可视化界面，为用户提供了一个完成操作指令的窗口。Activity 组件拥有其自身特有的生命周期与运行状态[35]。每个 App 都运行在自己的 Linux 进程中。当这个应用的某些代码需要执行时，进程就会被创建，并且将保持运行，当系统需要释放它所占用的内存时停止进程。Android 组件的生命周期是由系统控制，而非程序自身直接控制。图 2.2 为 Android 组件的生命周期整体图。

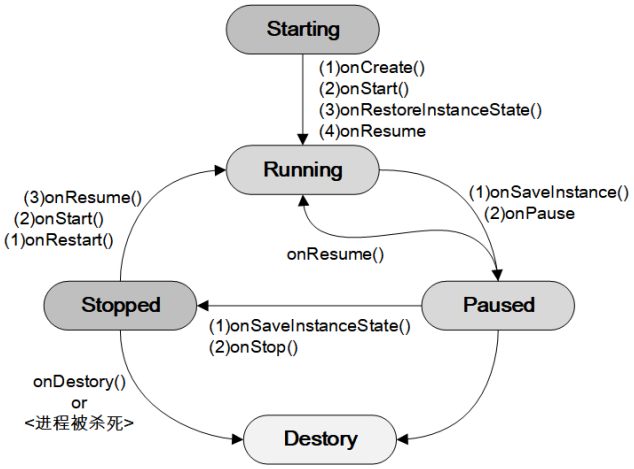


图 2.2 Android 组件的生命周期

（2）Service：Service 是运行在后台，并执行长时间操作的组件。Service 是在没有任何用户界面的情况下运行的应用程序组件，例如在后台播放音乐。

（3）Broadcast Receiver：Broadcast Receiver 简单地响应从系统或其他应用程序发来的广播消息。

（4）Content Provider：它的作用是将程序的内部数据和外部程序进行共享，并为数据提供外部访问接口。因此，每个应用程序都有一个 Content Provider，它相当于应用程序之间的结构化接口，访问彼此的数据。Content Provider 之间共享的数据由 SQLite 数据库或文件系统返回。

2.1.2 AndroidManifest.xml 文件

AndroidManifest.xml 是应用程序中权限声明和定义配置的文件，每个应用的根目录中都必须包含一个 AndroidManifest.xml 文件，我们可以通过该文件来获取组件的相关配置信息，其中描述了包中的组件、全局数据和各自的实现类等。表 2.1 为该文件中常见元素及其描述。

表 2.1 AndroidManifest.xml 文件常见元素及其描述

元素	描述
<manifest>	根节点，对包中所包含的内容进行描述
<uses-permission />	应用程序正常运行申请的权限，可以申请多个权限，即一个.xml 文件中可以有多个此元素
<permission />	声明一个安全授权，限定其他应用程序能否访问此程序中的资源，一个 xml 文件可以包含多个此元素
<instrumentation />	声明用于测试组件的代码，一个 xml 文件可以包含多个此元素
<uses-sdk />	用于声明当前应用程序能够兼容的最低版本号的 SDK
<application>	包中应用程序组件声明的根节点
<activity>	用户与应用程序交互的桥梁，当用户点击应用时，出现在手机上的主页面就是一个 activity
<intent-filter>	声明组件支持的 Intent 值
<action />	声明组件支持的 Intent Action
<category />	声明组件支持的 Intent Category
<data />	声明组件支持的数据类型或 URI
<meta-data />	一个 key-value，是父组件添加的数据，可以是 activity 等
<service>	运行在后台且可以运行任意时间
<receiver>	Intent Receiver 对应用程序读取的数据进行修改
<provider>	管理应用程序中所涉及的数据，并将数据传递给需要的组件

<manifest>和<application>元素必须存在且只能出现一次。大多数其它的元素可能会出现很多次或不出现。同一级别的元素通常不分先后顺序，例如，<activity>、<provider>和<service>元素可以按照任意的次序进行组合。但有两个例外：<activity-alias>必须紧跟在<activity>之后；<application>必须是<manifest>中的最后一个元素。

当创建应用时，需要通过 AndroidManifest.xml 文件声明应用程序的访问权限。如果组件定义了 android:permission 属性，则表示该组件在被调用的时候需要申请对应的权限，如果该权限的级别是 signature 或 signatureOrSystem，则该组件只能被具有相同签名的应用程序授权，这样该组件即便是暴露在外，也因为其他程序无法获得调用权限而不会受到攻击。在申请权限时也要注意，尽可能遵循“最小权限原则”，避免申请的权限过多而增加遭受攻击的风险。

2.2 Android 组件安全机制

2.2.1 Linux 系统内核安全机制

由于 Android 系统的底层使用了 Linux 系统内核，所以 Android 系统继承了 Linux 系统的权限访问机制并将其运用到 Android 的文件权限访问控制中。Android 系统中的每个文件都有九个访问权限控制位，这些访问控制位分三组，分别对应文件拥有者、用户组以及其他用户这三种用户对文件的三种操作权限。这三种操作权限分别为，对文件的读、写、执行权限。每个应用程序都有自己的数据存储目录，该目录的所有者是该应用程序的进程用户，不同的应用程序的进程用户不同，这样就达到了不同应用程序数据目录之间的权限隔离，恶意进程因为没有文件的操作权限而不能直接更改其他应用程序的文件。Android 系统每个应用程序的文件都被分配了对应的应用级用户，而系统级文件仅归系统或根用户所有，这样恶意应用就无法对系统文件进行攻击和破坏。

Android 系统的内存管理单元（Memory Management Unit ，MMU）具有高速缓存、虚拟地址映射等功能，为系统提供了硬件层的内存访问控制来检查是否存在内存的错误访问。MMU 为每个应用程序的进程分配了独立的地址空间，应用程序进程通过 MMU 来进行内存访问，使得一个应用程序的进程不能访问其他应用程序进程的内存页，或者侵占其他进程的内存空间，能够有效地防止其它进程对应用程序进程的内存进行恶意访问或破坏。

2.2.2 应用程序沙箱防御机制

一般来说，Android 应用程序使用 Java 语言编写，并通过 Android SDK 提供的编译工具编译成 dex 字节码，在 Dalvik 虚拟机上运行。Dalvik 虚拟机是 Google 专门为 Android 系统开发的 Java 虚拟机，它运行时占用的内存空间小，并且可以同时运行多个虚拟机实例。每个 Android 应用程序运行在单独的 Dalvik 虚拟机实例中，且每个 Dalvik 虚拟机实例对应一个单独的 Linux 进程，Android 系统通过这种方式实现了应用程序的安全沙箱机制，如图 2.3 为 Android 沙箱机制示意图。

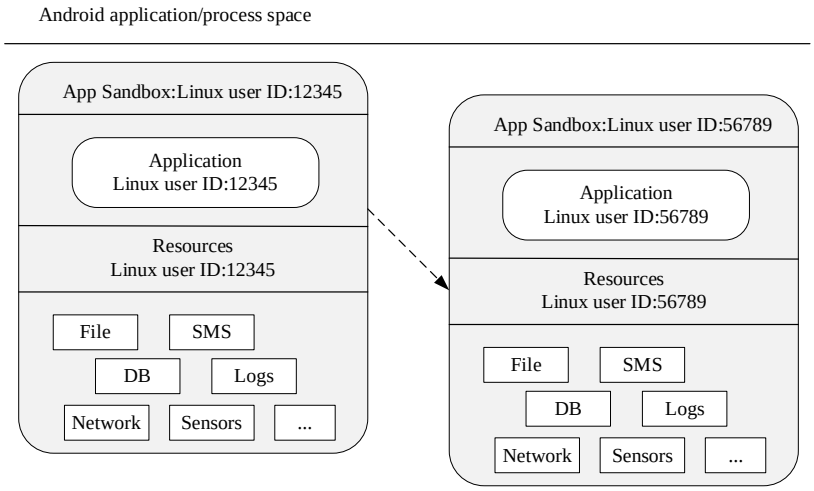


图 2.3 Android 沙箱机制示意图

2.2.3 权限机制

Android 系统使用权限机制[36]来进行强制访问控制，权限机制是通过限制对受限数据（如用户联系人信息、短信等）以及受限操作（如连接配对设备、拍照和录制音频等）的访问来保护用户隐私的一种隐私保护机制。一些和系统安全相关的操作在默认情况下不能被应用程序执行。应用程序操作权限分为四级，如表 2.2 所示。

表 2.2 Android 权限体系等级

权限级别	描述
普通级	该权限对应的操作不会对系统和数据造成危害，声明了此权限的应用程序，系统会默认授予此权限
危险级	该权限对应的操作能够危害到系统安全，声明了此权限的应用程序在安装时系统会给予授权提示
签名级	只有使用了相同的数字证书进行签名的应用程序才会被授予此权限
系统级	与签名级权限类似，只有和系统 ROM 使用了相同数字证书签名的应用程序才会被授予此权限

2.2.4 Intent 通信的安全措施

组件私有化：如果需要组件之间互相通信，则需要为组件设置<intent-filter>子标签，在这种情况下组件默认是暴露在系统环境中，可以接收来自其他程序的 Intent 消息。这时为了防止组件对外暴露，可以在 AndroidManifest.xml 文件中设置组件的 android：exported 标签为 false，声明该组件只能在内部使用，从而拒绝来自外部的 Intent 消息。

设置组件访问权限：如果要让组件参与跨应用调用，则可以给组件添加 android：permission 属性，声明调用该组件的应用需要的自定义访问权限。这样，只有申请了对应自定义权限的第三方应用才能调用该组件。

Android 四大组件之间都可以使用 Intent 消息对象来实现组件之间的通信。Intent 对象包含的属性如表 2.3 所示，可以在组件之间传递丰富的信息。

表 2.3 Intent 对象属性描述

属性	描述
Component Name	指明将要对 Intent 消息进行处理的组件
Action	字符串对象，设置该 Intent 会触发的操作类型
Data	描述了 Intent 的动作所能操作的数据的 URL 及 MIME 类型
Category	字符串对象，对执行操作的类型进行描述
Extras	Extras 部分是由 type、name、value 三个要素构成，能够向目标组件传递丰富的数据
Flags	指示 Android 系统将如何加载并运行的一个操作

Activity、Service、Broadcast Receiver 组件都可以通过 Intent 被其他组件所调用，对应的函数如表 2.4 所示。例如，如果要开启一个组件，可以通过 Intent 对象的 setClass 方法来设置要开启 Activity 组件的名称，调用 startActivity 方法来开启该 Activity 组件。

表 2.4 可以通过 Intent 启动的组件与应对方法

组件类型	方法
Activity	StartActivity() StartActivityForResult()
Service	startService() bindService()
Broadcast	sendBroadcast() sendOrderedBroadcast() sendStickyBroadcast()

2.3 模糊和组合测试方法

2.3.1 模糊测试方法

（1）概念

模糊测试最早在 1989 年由 Wisconsin-Madison 大学的 Barton Miller 教授提出，用于测试 UNIX 系统的健壮性。依照 Miller 等人的说法，一个模糊输入可能是没有预料到的输入，模糊测试就是一种通过提供非预期的输入并监视异常结果来发现漏洞的方法。文献[37]对模糊测试相关的术语进行了如下定义：

定义 2.1（**模糊化**）模糊化是使用从输入空间（“模糊输入空间”）采样的输入来执行被测程序。

定义 2.2（**模糊测试**）模糊测试是使用模糊化来测试一个程序是否违反了正确性策略。

定义 2.3（**模糊程序**）模糊程序是在被测程序上执行模糊测试的程序。

定义 2.4（**模糊活动**）模糊活动是在具有特定正确性策略的被测程序上执行模糊程序，也就是对被测程序执行模糊测试的过程。

定义 2.5（**故障程序**）故障程序是模糊程序的一部分，它用来确定被测程序的执行是否违反了特定的正确性策略。

（2）流程

尽管模糊测试侧重于发现违反策略的行为，但它所基于的技术可以应用于其他问题。本文将使用模糊测试的方法设计生成测试用例并进行实验。模糊测试的一般工作流程[38]如图 2.4 所示。

图 2.4 模糊测试一般流程

第一，选择模糊测试目标。第二，识别输入。模糊测试可以对内部应用程序进行安全测试，也可以对第三方应用程序进行漏洞挖掘。第三，构造测试用例。这个过程中需要根据应用程序具体的运行情况采用预定义或者变异的方式生成测试用例。该阶段是模糊测试过程中最为重要的部分。第四，执行测试用例。此阶段主要是将构造好的测试用例作为信息输入源的数据发送到应用中。第五，监测系统是否异常。监测过程十分重要，该阶段主要存在两种情况：成功和失败。测试成功表明发送到被测系统中的测试用例数据触发了漏洞，而测试失败则可以根据应用程序的不同响应做出相对应的调整，继续模糊测试行为。

模糊测试与其他的一些如代码审计、静态分析[39]方式相比存在许多优点。第一，模糊测试在不需要程序源代码的情况下便能够发现程序中的错误；第二，模糊测试不会因为被测应用内部的复杂性而受限制；第三，可复用性较强，一个测试用例可以适用于多种程序[40]。然而，在构造测试用例时，现有模糊测试框架普遍存在测试用例单一、单维的问题。针对这样的问题，本文采用基于生成和基于变异的相结合的策略作为模糊测试用例构造策略。

2.3.2 组合测试方法

为了检测软件在多种因素作用下的运行状态，必须要设计相应的测试用例来测试软件运行过程中出现的多种问题[41]。组合测试是一种科学有效的软件测试方法，其目的在于利用较少的测试用例实现软件运行测试[42]。组合测试可以分为以下几个阶段：建立组合测试模型、测试用例生成、测试执行、故障定位、进一步测试及回归测试和测试评估，

如图 2.5 所示。

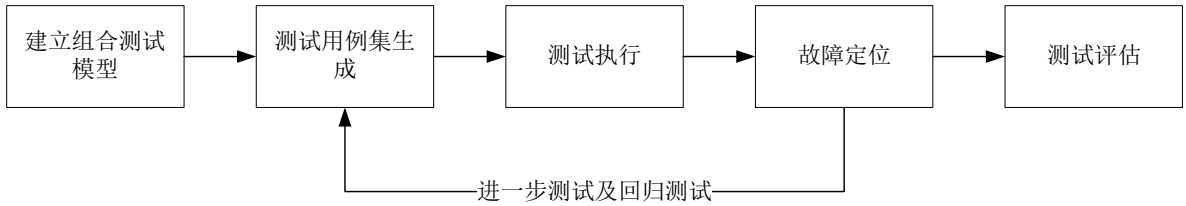


图 2.5 组合测试基本流程

组合测试的关键问题之一就是要从庞大的参数组合空间中挑选少量的测试用例来实现对程序科学有效的测试，即组合测试的测试用例生成问题，此问题是一个 NP 难问题。表 2.5 为组合测试用例生成考虑的因素。

表 2.5 组合测试用例生成考虑的因素

生成因素	测试用例集类型	指定测试用例	解决约束	生成技术
组合测试用例生成过程中组合有 5×2×2×5=100 种	正交表	指定部分测试用例作为种子	解决约束	贪心算法
	K 维覆盖表		不解决	数学方法
	可变力度覆盖表	不指定		启发式演化算法
	增量覆盖表			随机算法
	其他组合设计			方法组合

2.4 本章小结

本章首先介绍了 Android 基本组件和 AndroidManifest.xml 文件，然后阐述了 Android 组件安全机制，包括 Linux 系统内核安全机制、应用程序沙箱防御机制、权限机制和 Intent 通信的安全措施。最后描述了本文中使用的模糊测试和组合测试方法。

3 安卓 App 配置规则研究

本章针对 Android 官方文档中部分组件配置描述语义模糊的问题，用模糊测试和组合测试方法构造测试用例，通过对用例运行结果的分析，明确这些配置项的含义，在此基础上进一步采用一阶谓词逻辑描述这些配置项的语义，形成了 32 条形式化配置规则，以供设计及开发人员使用，进而减少设计漏洞。

3.1 问题分析

当代码缺少注释[43]时，开发人员可以参考 Android 官方文档、咨询团队成员以及在互联网中查找获取相关信息。但通过以上方式开发人员不一定能够获取代码的完整信息，也不能准确理解代码中配置的明确含义。尽管 Android 官方文档描述了组件配置元素，但是一些描述没有给出准确的语义定义，一些描述没有给出相关配置元素之间准确的关系，因此这些自然语言描述存在不确定性、二义性，会造成开发者一些理解性的偏差，可能致使开发者对 App 配置不当，产生安全漏洞。下面通过两个例子来说明，Android 官方文档在用自然语言描述配置的含义时存在二义性：

（1）每一个组件都可以通过<permission>标签配置 enforced permission，该权限说明了此 App 施加给与其通信时其他 App 应该具有的权限，但 Android 官方文档中并没有对每个组件配置该类权限的数量进行明确的约束。测试实验表明只能配置一个 enforced permission。

（2）根据 Android 官方文档，可以在 AndroidManifest.xml 文件中设置应用程序或组件的进程（process）属性，但是对于该属性值应遵守的格式没有给出严格说明。根据测试实验结果，可以得出 process 属性值的格式必须遵守 Java 包格式，否则安装该应用程序时会出现错误。

本章通过模糊测试和组合测试方法构造测试用例，模拟各种可能的配置以及不同配置项之间的组合关系，通过开发应用程序构造若干测试用例，运行测试用例并分析测试结果，将运行结果与 Android 官方文档中的描述结合后，确认 Android 应用程序配置项的准确含义以及配置项之间的关系。

3.2 测试用例的构造

3.2.1 基于模糊测试的测试用例构造方法

模糊测试的关键步骤是构造测试用例，现阶段主流的构造策略是基于生成的策略和基于变异的策略。基于生成的策略，是指通过分析被测目标输入变量的信息，依据测试

经验生成，构造正常或非正常的测试用例，如图 3.1 所示。基于变异的策略，是指在分析已知样本的基础上，使用随机变异或启发式方法修改测试样本的某些输入字段，该策略不需了解被测目标的相关背景知识即可生成测试用例。其优点在于不需要大量的前期工作，仅需要提供一个或多个变异样本。

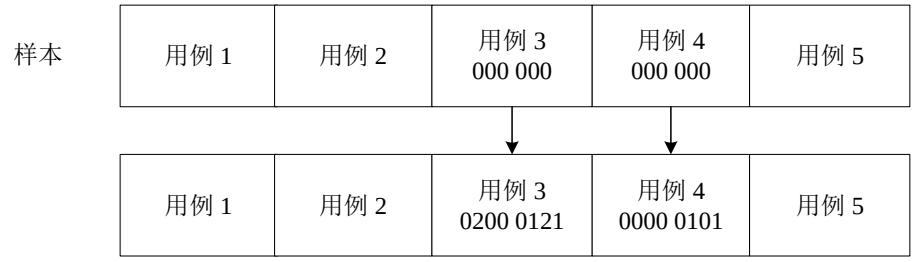


图 3.1 构造测试用例策略

模糊测试方法在构造测试用例时，现有模糊测试框架中普遍存在测试用例单一、单维的问题。本节测试用例策略为基于生成和基于变异相结合的构造策略。其中，基于生成的策略需要依据测试经验生成测试用例，基于变异的策略则是采用字段变异的手段构造测试用例。本节构造测试用例的步骤为：首先分析待测配置的基础字段的格式，依据测试经验确定可变异字段，然后，通过选取可变异字段集合中的有效字段取值替换基础字段中相应的可变异字段取值，来生成不同的测试用例，使这些测试用例进入模糊活动，最后通过控制可变异字段取值的数量来控制生成测试用例的数量，防止无效测试用例的生成。

下面通过一个例子来说明使用本节构造策略的过程。该案例为测试 SharedUserId 属性值的正确格式。根据开发实践中的一般情况，SharedUserId 的属性值类型为字符串，因此首先定义格式为“xxx.xxx.xxx”的字符串作为基础字段，然后在基础字段上进行变异，以构造测试用例，最后运行用例并分析其测试结果得到 SharedUserId 属性值的正确格式。

如图 3.2 所示，我们设定该案例的基础字段为“abc.bbc.ccd”，然后确定该基础字段的可变异字段，每个变异字段的取值方式有很多：（1）数字与字母的组合，如用例“bc#.12n.rm5”中的变异字段“12n”;（2）数字与特殊符号的组合，如用例“67#.1@^.nde”中的变异字段“67#”;（3）字母与特殊符号的组合，如用例 “ab*.ddd.**&1”中的变异字段“ab*”;（4）数字、字母和特殊符号的组合，如用例“kk6.9s@.nde”中的变异字段“9s@”。最后，通过可变异字段的不同取值得到若干测试用例，开发应用程序，并分析程序运行结果。若某条测试用例的测试结果为 App 能够正常安装和运行，则该条测试用例的取值方式为 SharedUserId 属性值的正确格式。

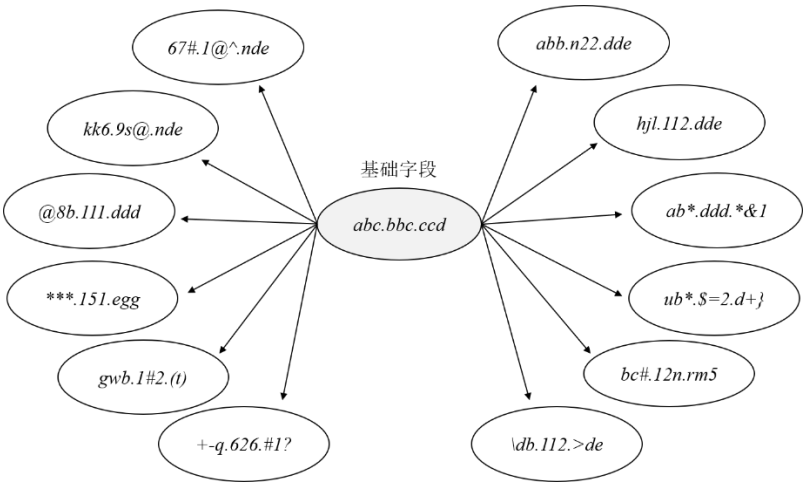


图 3.2 sharedUserId 取值方式

3.2.2 基于组合测试的用例构造方法

基于组合测试的用例构造方法就是通过对一个问题中涉及到的n个配置项进行各种不同的组合，来构造测试用例。其中，对于任意t（通常$2 \le t \le n$较小的整数）个参数的所有可能的组合至少被一个测试用例覆盖。

下面通过一个案例来说明使用组合测试方法构造测试用例的过程。该案例为测试 multiprocess 属性的语义，以及 multiprocess 属性值与 process 属性值之间的关系。案例中涉及 3 个配置项：应用程序的签名、process 属性、multiprocess 属性。（1）应用程序的签名有两种情况：两个应用程序签名相同；两个应用程序签名不同。（2）process 属性有两种情况：process 设值；process 未设值。（3）multiprocess 属性有两种情况：multiprocess 设置为 true；multiprocess 设置为 false。本节基于 3 个配置项不同取值的三三组合构造测试用例，部分用例如表 3.1 所示。

测试前提：App_1和App_2分别配置了 3 个 Activity：FirstActivity、SecondActivity 和 ThirdActivity，且App_1中的任意 Activity 都可以启动App_2中的任意 Activity。将App_1的 SecondActivity 进程名设置为“:second”，将App_1的 ThirdActivity 进程设置为“:third”。

执行测试用例 1 中的路径（1），点击App_1的 MainActivity 按钮，跳转至App_1的 SecondActivity，再点击 SecondActivity 按钮，跳转至App_2的 ThirdActivity，测试结果显示，此时App_1.SecondActivity 进程名为 com.demo.app1:second；App_2.ThirdActivity 进程名为 com.demo. app1:second。然后执行路径（2），结果为App_1.ThirdActivity 的进程名为 com.demo.app1:third；App_2. ThirdActivity 的进程名为 com.demo. app1:third。通过分析App_2.ThirdActivity 在路径（1）和路径（2）中的进程名可知，该 Activity 运行在启动它的 Activity 的进程中。由此可得，当两个应用程序的签名相同，process 属性值未设置，

mutiprocesss=true 时，该 Activity 可以运行在启动它的 Activity 进程中。执行测试用例 2，可得当两个 App 签名不同，process 取值，multiprocess=false 时，Activity 被启动时与启动它的 Activity 进程无关，总是运行在默认进程中。

表 3.1 测试用例说明

ID	测试用例	测试结果（查看进程名）
1	当两个 App 的签名相同，process 属性值未设置，mutiprocesss=true 时，通过不同路径启动 Activity： （1）App_1.MainActivity→App_1.SecondActivity→App2.ThirdActivity （2）App_1.MainActivity→App_1.ThirdActivity→App_2.ThirdActivity	（1）App_1.SecondActivity 进程名：com.demo.app1:second App_2.ThirdActivity 进程名：com.demo.app1:second （2）App_1.ThirdActivity 进程名：com.demo.app1:third App_2 .ThirdActivity 进程名：com.demo. app1:third
2	当两个 App 签名不同，process=“:remote”，multiprocess=false 时，通过不同路径启动 Activity： （1）App_1.MainActivity→App_1.SecondActivity→App2.SecondActivity （2）App_1.MainActivity→App_1.ThirdActivity→App_2. SecondActivity	（1）App_1.SecondActivity 进程名：com.demo.app1:second App_2 .SecondActivity 进程名：com.demo.app2:remote2 （2）App_1.ThirdActivity 进程名：com.demo.app1:third App_2 .SecondActivity 进程名：com.demo.app2:remote2

3.3 测试对象与结果

本节展示了本章测试对象类型（共五种类型）、测试对象以及通过分析测试结果得出的结论。第 3.4 节使用一阶逻辑谓词将以下所得结论进行形式化描述。

（1）应用程序共享用户 ID

在应用程序的基本属性配置中，最复杂的是共享用户 ID 的应用程序配置。表 3.2 描述了本章测试应用程序共享用户 ID 相关内容的测试对象及结论。

表 3.2 应用程序共享用户 ID

类型	测试对象	结论
应用程序共享用户 ID	sharedUserId 属性值的正确格式	设置 sharedUserID 属性值时，必须遵守 Java 包格式
	多个 App 共享用户 ID	两个及两个以上 App 共享用户 ID 的条件是它们的签名与 sharedUserID 取值都相同
	sharedUserId 值为另一个已安装的包名	不能直接将一个 App 的 sharedUserID 设置为另一个已经安装的 App 包名，这样不可以实现用户共享
	应用程序包的覆盖	两个 App 包名相同，签名不同时，第二个安装的 App 在安装过程中会出错；当包名、签名均相同时，第二个安装的 App 在安装过程中会覆盖先前安装的 App

	定义但不注册组件	定义一个组件后，必须在 AndroidManifest.xml 中注册，否则不可以使用
	含有多个 MainActivity	一个应用程序可以定义多个 MainActivity，但以第一个“MainActivity”作为主活动

（2）应用程序及组件进程（process）分析

使用 process 属性可以指定应用程序或者其组件的进程，如果不设置 process 属性，那么 Activity 启动后，默认运行在当前应用程序进程中。如果设置 process 属性，在一定条件下，可以把 Activity 组件部署到其他进程中。表 3.3 描述了本章测试应用程序及组件进程相关内容的测试对象及结果。

表 3.3 应用程序及组件进程分析

类型	测试对象	结论
应用程序及组件进程分析	process 的正确格式	设置应用程序的 process 属性值时，必须遵守 Java 包格式；未设置 process 属性时，应用程序进程名默认为项目包名
	应用程序及其组件的优先级	当同时在应用程序和组件设置 process 属性时，组件优先级更高
	将组件部署到其它进程	两个不同 App 中的组件运行于同一个进程，两个应用程序需要同时满足以下三个条件，相同的 sharedUserId、相同的 keystore 进行签名以及为组件设置相同的 process 值
	exported 的含义及其缺省值	exported 默认值为 false，即该组件不可以被外部访问
	multiprocess 属性与组件实例化分析	签名不同的两个 App 的组件，无论它们的 process 为何值，无论 multiprocess 为何值，组件被启动时都与启动组件的进程无关，总是运行在默认进程中
		签名相同的两个 App 的组件，无论 process 设置与否，当 multiprocess 设置为“false”时，该组件总是运行在默认 App 进程中；当 multiprocess 为设置“true”时，该组件可以运行在启动它的组件进程中

（3）应用程序和组件权限（permission）配置研究

我们可以使用<uses-permission>声明 App 具有的权限，也可以使用 permission 属性对 App 的使用者做限制。表 3.4 描述了本章测试应用程序和组件权限配置相关内容的测试对象及结论。

表 3.4 应用程序和组件权限配置研究

类型	测试对象	结论
应用程序和组件权限配置研究	共享用户间 App 的权限集合	App 中的 packages.xml 文件中只记录普通权限，不记录危险权限。危险权限在运行时动态获取
		在 App_1 已安装的条件下安装 App_2，查看 App_1 中的 packages.xml 文件记录的权限，结果为取两个 App 普通权限的并集，且不记录危险权限，危险权限需要用户在 Android 设备的权限管理中手动授予
	enforced 权限的继承	如果应用程序请求系统授予其危险权限，而此时应用程序在同一权限组中已存在另一项危险权限，则系统会立即授予该权限，而无需与用户进行交互
		每一个组件都可以通过<permission>标签配置 enforced 权限，该权限说明了此 App 授予与它通信时其它 App 应该具有的权限，实验表明每一个组件只能配置一个 enforced 权限

（4）组件间通信的权限分析

Intent 在 Android 中是一个十分重要的组件，它是连接不同应用的桥梁和纽带。Intent 的使用分为两个方面，一个是发出 Intent，另一个则是接收 Intent。表 3.5 描述了本章测试组件间通信的权限相关内容的测试对象及结论。

表 3.5 组件间通信的权限分析

类型	测试对象	结论
组件间通信的权限分析	IntentFilter 与 exported 属性分析	跨应用启动 Activity 需要设置 exported 属性为“true”，如果定义了至少一个 IntentFilter，那么 exported 的默认值为 true，如果没有定义 IntentFilter，则 exported 默认值为 false
	由 Intent action 所施加的权限	permission 要求的权限与 action 间接需要的权限不同，permission 属性值是要求发送者需要具有的权限，action 则是应用本身访问此 action 需要具有的权限，即自身需要具有的权限

（5）四种 LaunchMode

Android 中有四种 LaunchMode 启动模式，分别是 Standard、SingleTop、SingleTask、SingleInstance。表 3.6 描述了本章测试四种 LaunchMode 相关内容的对象及结论。

表 3.6 四种 LaunchMode 分析

类型	测试对象	结论
LaunchMode	Standard 模式	标准模式下，每次启动一个 Activity 都会重新创建一个新的实例，不管这个实例是否存在。在这种模式下谁启动了这个 Activity，那么这个 Activity 就运行在启动它的 Activity 所在的栈中
	SingleTop 模式	该模式下，如果新建的 Activity 已经位于任务栈的栈顶，那么 Activity 不会被创建，同时它的 onNewIntent 方法会

		被回调。如果新的 Activity 实例已存在但是没有位于栈顶，那么新的 Activity 实例仍然会重新创建
	SingleTask 模式	该模式下，只要 Activity 在一个任务栈中存在，那么多次启动此 Activity 都不会重新创建实例，当 taskAffinity 和 SingleTask 配合使用时，taskAffinity 设置了任务栈的名称，当不存在 taskAffinity 指定的任务栈时，应用程序会先创建任务栈，再将 Activity 放到新建的任务栈中
	SingleInstance 模式	该模式下，Activity 只能单独位于一个任务栈中。当 Activity 启动时应用程序会为它新建一个任务栈，它单独在这个新的任务栈中，后续的请求均不会创建新实例，且后面启动的 Acivity 均不会放置在此任务栈中

3.4 配置规则的形成

3.4.1 形式化模型的建立

Android 应用程序的配置属性可以从应用程序的静态、动态特性进行研究。在本节中，使用 E-R（实体-关系）表示法来描述每个基本概念，即每个概念都被描述为一个属性元组，每个属性元组都受到特定类型的限制。如表 3.7 为约束属性的基本类型，基于基本类型，然后定义配置元素的概念和约束关系。

表 3.7 应用程序的属性类型及解释

集合名称	解释
JNANE $JNAME_{\perp}$	使用 Java 包格式命名字符串，用于标记应用程序、组件或权限 $JNAME_{\perp}$是使用底部元素扩展JNANE的集合, 即$JNAME_{\perp} = JNAME \cup \{\perp\}$. 如果变量采用“⊥”作为值, 这意味着其值不存在或未定义
PKGNAME $PKANAME_{\perp}$	包或组件名称的有限集合。用于在特定上下文中标记应用程序或组件。 $PKGNAME_{\perp} = PKGNAME \cup \{\perp\}$.如果变量为“⊥” 作为值，这意味着该值不存在或未定义
USRNAME	用户名的集合。Android 系统中的用户名是一个前缀为“u0_a”的字符串
PROCNAME	进程名称的集合
ID $ID_{\perp}$	标识符的集合。每个标识都是一个正整数，通常由 Android 系统生成，来表示组件的用户 Id、进程 Id 或实例 Id。类似的，$ID_{\perp}$是使用底部元素扩展 ID 的集合，即$ID_{\perp} = ID \cup \{\perp\}$
PERMNAME	Android 系统预定义的权限集或用户根据特殊需要定义的权限集。例如，android.permission.CALL_PHONE 是一种允许应用程序拨出电话而无需用户在拨号界面上确认的权限
URITYPE	URI 的集合。URI 为标识特定资源的字符串

ACTION	action 的集合。action 名称取值遵循 Java 包格式
INTENT	一组有限的 Intent 集合
APISET	Android 系统提供的有限 API 集合
INSTANCES	创建的总实例集
2^{PERM}	由所有PERM的子集所构成的集合

具有静态特性的配置元素指定了一个应用程序的类型或结构属性，可以直接从应用程序的配置文件和代码中提取。具有动态特性的配置元素与应用程序运行时的行为或创建的实例有关。我们形式化定义了具有静态、动态特性的配置元素，同时对这些配置元素的强制执行的语法进行了定义。本节使用如下四个例子进行说明，其中具有静态特性的配置元素为 Application、Activity，具有动态特性的配置元素为 Intent、Task。

定义 3.1 (**Application**) Application 定义为一个七元组:

$$App = def < name, uid, cert, sharedUserId, usedPerm, enfPerm, process >$$

其中：

- $name \in JNAME$: 应用程序的包名，用于标识安装在移动设备中的不同应用程序。
- $uid \in ID$: Android 系统在安装应用程序期间分配给应用程序的唯一用户标识符。
- $cert$: 由应用程序开发人员签署的证书，用于保护应用程序不被恶意开发人员篡改。
- $sharedUserId \in JNAME_{\perp}$: 应用程序的共享用户名，在应用程序的 AndroidManifest.xml 文件中声明。共享用户名相同的应用程序可以视为一个组，同一组中成员可以自由地与其他成员的组件交互。
- $usedPerm \subseteq PERM$: 应用程序请求的权限集。Android 系统将所有请求的权限授予应用程序，而无需任何验证。
- $enfPerm \in PERM$: 一个应用程序施加给与其通信时其他应用程序应该具有的权限。
- $process \in \in PRONAME_{\perp}$: 进程的名称。进程包括运行中的程序和程序使用到的内存和系统资源。

定义 3.2 (**Activity**) Activity 定义为一个九元组：

$$Activity = def < parent, name, exported, multiprocess, launchMode, enfPerm, main，taskAffinity，procname >$$

其中：

- $parent \in APP$: 一个 Activity 所属的应用程序。
- $name \in JNAME$: Activity 的名称。

- $exported \in BOOL$: Activity 是否可以由其他应用程序的组件调用，Activity 允许被启动取值为“true”，不允许则为“false”。
- $multiprocess \in BOOL$: Activity 在启动它的组件的进程中能否被实例化，Activity 可以多实例为“true”，不可以则为“false”。
- $launchMode \in \{standard, singleTop, singleTask, singleInstance\}$: Activity 的启动模式，它涉及如何实例化 Activity（Activity 的单个或多个实例），以及如何在任务栈中管理 Activity 实例。
- $enfPerm \in PERM$：对 Activity 的强制权限。此权限由 Activity 对与 Activity 交互的其他应用程序的组件强制执行。
- $main \in BOOL$: 表明 Activity 是否为 MainActivity。
- $taskAffinity \in JNAME$: 将在其中分配 Activity 实例的任务名称。
- $process \in JNAME$: 运行 Activity 组件的进程名称。设置此属性允许 Activity 在与其所在应用程序分离的进程中运行。它的分配将覆盖其父应用程序的分配。

定义 3.3 (Intent) Intent 定义为一个六元组：

$$Intent = def < sender, action, data, recvr, sendPerm, recvPerm >$$

其中：

- $sender \in INSTANCES$: 通过特定方式调用并发送 Intent 组件实例。
- $action \in ACTION$: 由 Intent Receivers 执行的动作，例如 ACTION_VIEW 等。
- $data \subseteq URI$: 需要操作的数据，其格式应与操作匹配，以形成适当的操作/数据对。
- $recvr \in iACT \cup iSERV \cup iBROADCAST$: Intent 的接收者。通常，一个 Intent 只能由一个组件接收和处理，但对于 Broadcast Receiver，如果多个 Receiver 的 Intent Filter 相互匹配，则允许多个 Receiver 接收一个 Intent。
- $sendPerm \in 2^{PERM}$: 对 Intent 发送人强制执行的权限集。
- $recvPerm \in 2^{PERM}$: 对 Intent 接收者强制执行的权限集。

定义 3.4 (Task) Task 定义为一个三元组:

$$task = (tid, state, state)$$

- tid: Task 的 ID。每个 Task 都有一个唯一 ID。
- $stack$: 任务栈，Activity 按照各自的打开顺序排列在栈中。
- $state \in \{\text{FORWGROUND, BACKGROUND}\}$: 任务的状态。FOREGROUND 表示任务在主屏幕显示，BACKGROUND 表示任务在后台。任一时刻，只有一个 Task 处于 FOREGROUND 状态。

除上述四个形式化定义，我们还定义了具有静态特性的配置元素 Service、Content

Provider、Broadcast Receiver、Permission，和具有动态特性的配置元素 Process、Activity Instance。

3.4.2 配置规则

建立配置项形式化模型后，使用一阶谓词逻辑对第 3.3 节所有测试结论给予形式化描述，从而形成配置规则。下面对其中三类测试对象得出的规则进行举例说明。

（1）应用程序的基本属性配置

谓词 1-1: $app1, app2 \in APP$，使用谓词$sharedUserId$（$app1, app2$）表示这两个配置为共享同一用户：

$$sharedUserId(app1, app2) = app1.sharedUserId = app2.sharedUserId \wedge app1.cert = app2.cert$$

谓词 1-2: $inSameProcess(app1, app2)$: 表示 app1 和 app2 在同一进程中运行.

谓词 1-3: $updatedBy(app1, app2)$: 表示 app1 已由 app2 成功更新.

规则 1-1:共享相同用户 ID 的应用程序必须具有相同的证书：

$$\forall app1, app2 \in APP \bullet sharedUserId(app1, app2) \Rightarrow identical(app1.cert, app2.cert)$$

规则 1-2:当且仅当两个应用程序共享相同的用户 ID 并配置为相同的进程名称时，它们才能在同一个进程中运行。

$$\forall app1, app2 \in APP \bullet sharedUserId(app1, app2) \wedge app1.process = app2.process \Leftrightarrow inSameProcess(app1, app2)$$

规则 1-3: App 更新条件:如果用 app2 更新 app1，那么两个 app 的名字与签名必须相同。

$$\forall app1 \in APP, app2 \in APPD \bullet updatedBy(app1, app2) \Rightarrow app1.name = app2.name \wedge app1.cert = app2.cert$$

规则 1-4:如果两个或两个以上应用程序具有相同的用户 ID，那么它们必须同时有相同的 sharedUserId 值和 cert 值，即 cert 相同与 sharedUserId 相同互为充要条件。

$$\forall app1, app2 \in APP \bullet sharedUserId(app1, app2) \Leftrightarrow app1.uid = app2.uid$$

规则 1-5:一个应用程序只有一个 MainActivity。

$$\forall app \in APP \bullet \exists 1 act \in ACTIVITY \bullet act.main = true \wedge act.parent = app$$

（2）应用程序及组件进程分析

谓词 2-1:$inSameprocess(app1, app2)$: 两个不同 App 中的组件运行于同一个进程。

谓词 2-2:$inSameUser(comp1, comp2)$: 组件 1 和组件 2 在同一进程中运行。

谓词 2-3: $inSameKeystore(app1, app2)$: app1 和 app2 使用相同 keystore 进行签名。

规则 2-1:使两个不同 App 中的组件运行于同一个进程，有以下三个条件：两个应用程序使用相同的 sharedUserId，两个应用使用相同的 keystore 进行签名，为组件

设置相同的 process。

$$\forall app1, app2 \in APP \bullet \exists\, inSameProcess(app1, app2) \Rightarrow sharedUserId(app1, app2) \vee inSameKeystore(app1, app2) \vee inSameUser(comp1, comp2)$$

规则 2-2:应用程序可以部署到默认进程以外的进程，前提是两个应用程序共享相同的用户 ID。

$$\forall app \in APP \bullet app.procName \neq app.name \Rightarrow (\exists\, app1 \in APP \bullet app.procName = app1.name \wedge sharedUserId(app, app1)$$

规则 2-3:如果两个应用程序共享相同的用户 ID，则一个应用程序的活动可以在另一个应用程序的进程中运行。

$$\forall act \in ACTIVITY, app \in APP \bullet act\ run\ in\ the\ process\ of\ app \Rightarrow sharedUserId(act.app, app) \wedge act.process = app.process$$

（3）组件间通信的权限分析

规则 3-1:如果 Intent 的发送方和接收方在不同的应用程序包中定义，而没有共享用户 ID，则必须将组件开放给接收方，并且发送方必须具有接收方施加给他的强制执行的权限。

$$\forall intent \in INTENT \bullet intent\ is\ sent\ successfully \Rightarrow intent.receiver.exported = true \wedge intent.receiver.enfPerm \subseteq intent.sender.type.usedPerm$$

规则 3-2:如果 Intent 通信发生在同一个应用程序或具有共享用户 ID 的应用程序内，则无需检查发送方的权限和接收方被强制执行的权限是否一致:

$$\forall intent \in INTENT \bullet (samePkg(intent.sender.type, intent.receiver) \vee sharedUserId(intent.sender.type.parent, intent.receiver.type.parent)) \Rightarrow sender's\ permission\ does\ not\ need\ to\ be\ checked.$$

规则 3-3:只要在应用程序包中定义了 Intent 的发送方和接收方，必须检查 Intent 的 sendPerm 和 recvPerm。

$$\forall intent \in INTENT \bullet intent\ is\ sent\ successfully \Rightarrow intent.sendPerm \subseteq intent.sender.type.usedPerm \wedge intent.recvPerm \subseteq intent.receivers.usedPerm$$

规则 3-4 如果缺少 exported 的值，则 Intent Filters 中可以包含其默认值*:*

$$\forall \text{svc} \in SERVICE \bullet (svc.intFilter = \Phi \Rightarrow \text{svc.exported} = \text{false} \vee \text{svc.intFiler} \neq \Phi \Rightarrow \text{svc.esported} = \text{true})$$

除以上 12 条配置规则外，我们基于表 3.2 到表 3.6 中的测试结论，得出了 32 条配置规则，供开发及设计人员使用，进而减少设计漏洞。

3.5 本章小结

本章首先详细阐述了模糊测试以及组合测试方法构造测试用例的过程，然后通过运行测试用例得出测试结果，分析了测试结果并得出测试结论，举例说明了配置项元素的形式化描述，最后使用一阶逻辑谓词描述了测试结论得出 32 条配置规则，并对规则进行了举例说明。

4 任务劫持攻击方法研究

由于开发人员对 Android 配置不当造成配置漏洞或者攻击者故意修改恶意应用程序有关配置属性时，可能会造成一些恶意攻击，比如欺骗攻击、网络钓鱼攻击等。本章为了展示由于配置方面的漏洞所引发的应用程序安全问题，以三种典型的任务劫持攻击为例，研究配置方面的脆弱性，实现了五组攻击实例，其中包括网络钓鱼攻击（2 组实例），欺骗攻击（1 组实例），勒索 App（2 组实例），然后统计了各大应用厂商中应用程序使用任务劫持条件的比例，最后给出了缓解此类攻击的安全指南。

4.1 问题分析

虽然 Android 系统的多任务机制提供了丰富的功能来增强用户体验，也为开发者在开发应用程序过程中提供了极大的灵活性，但该机制仍然存在严重的安全漏洞，可能引发任务劫持攻击。例如，攻击者可以通过配置 taskAffinity 和 allowTaskReparenting 等属性值，成功骗过系统并启动虚假界面，当用户在这个假界面中输入他们的隐私信息时，攻击者会立即得到这些信息，随后可以登录并控制那些应用程序。为了了解具体的任务劫持攻击过程，需要系统的研究 Android 多任务机制的行为模式。本章通过将这种行为模式投射到状态转换模型（威胁模型）中来分析任务劫持攻击的过程。

4.2 威胁模型

本节通过一个威胁模型来分析 Android 多任务机制下可能出现的任务劫持状态转换过程。下面本节从任务劫持攻击的攻击面、攻击向量和威胁模型三个方面对该类攻击进行描述。

攻击面是任务（Task）和任务栈（Back Stack）之间存在的特性。在 Android 中，任务是一些相关联的 Activity 集合，这些集合保存在任务栈中，并按照访问 Activity 的时间排序，用户单击 Back 按钮，将返回到当前任务中的最新 Activity 中。任务栈中的 Activity 可能来自相同或不同的应用程序，因此一个应用程序和恶意应用程序的 Activity 可以出现在同一任务栈中，这个应用程序可能会遭到任务劫持攻击。

攻击向量是指任务劫持攻击过程。例如，每当用户启动一个应用程序，攻击者可以通过调节恶意应用的配置属性，在用户无法察觉时，使恶意应用在它的控制下向用户显示一个伪造的界面，覆盖原始应用程序中的真实界面。

威胁模型，也就是劫持状态转换模型。系统中会存在多种任务状态转换过程，并且该过程可能比较复杂，在复杂的任务状态转换期间可能出现多种方式的任务劫持攻击，本节将劫持状态转换过程表示为一个威胁模型，该模型展示了攻击者如何操纵任务状态

转换过程，对应用程序实施任务劫持攻击。

我们对图 4.1 的威胁模型进行说明，该模型展示了欺骗攻击的任务状态转换。图 4.1 中 A 为系统默认情况下的任务状态转换，B 为任务劫持状态转换，Mal-root 为初始界面，Mal-main 为恶意应用程序伪造的界面，该界面模仿应用程序中的 Main 界面。在初始状态s_0中，用户显示的界面为应用程序的主界面，此时恶意软件任务在后台等待，当用户启动应用程序时，默认发生状态 A，即创建一个新任务，并在屏幕上显示应用程序的 MainActivity。但是，如状态转换 B 所示，恶意软件通过修改相关配置属性，将 Mal-main 从后台置于应用程序任务栈的栈顶，使用户界面显示 Mal-main。此时，用户无法察觉当前界面为伪造的界面，因为原始界面并没有显示在屏幕上。通过这种方式，Main 从启动时起就被恶意软件 Activity 顺利劫持，并且此任务中的所有用户行为目前都处于恶意软件的控制之下。

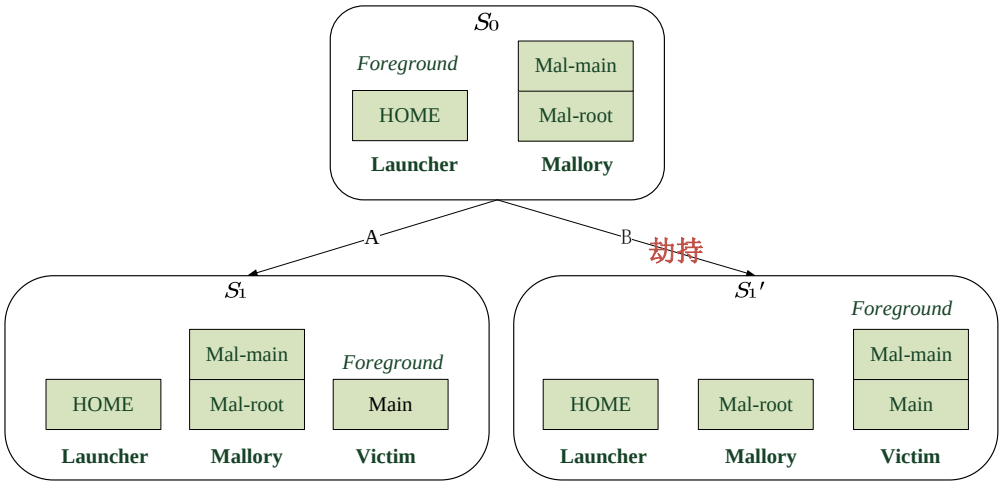


图 4.1 欺骗攻击的威胁模型

4.3 任务劫持条件

Android 系统提供了一组丰富的任务控制配置项，这些配置项使开发人员能更灵活地开发应用程序，例如将现有 Activity 置于另一个任务栈中，但同时这些配置项也可被攻击者利用。所有配置项的不同组合可以使任务状态转换更加复杂，从而导致多种方式的任务劫持攻击，因此，本章将这些配置项称为任务劫持条件。表 4.1 列出了 Activity 属性、Intent 标志、回调函数和 APIs 四种任务劫持条件。

表 4.1 Android 提供的可配置的任务劫持条件

Intent 标志 (FLAG_ACTIVITY_ *)	Activity 属性	回调函数	APIs
NEW_TASK SINGLE_TOP CLEAR_TOP REORDER_TO_FRONT NO_HISTORY CLEAR_TASK NEW_DOCUMENT(API 21) MULTIPLE_TASK	launchMode allowTaskReparenting taskAffinity allowTaskReparenting documentLaunchMode(API 21) finishOnTaskLaunch	onBackPressed()	startActivity() startActivities() TaskStackBuilder -class

4.4 模拟实验

本节将模拟在 Android 系统中劫持状态转换期间可能发生的三种任务劫持攻击。状态数量过多会导致任务状态转换过于复杂，为了更直观的展示劫持攻击过程，将任务状态的数量限制在引发特定攻击的情况下，为此，实验中定义了两个约束规则来控制状态数量。

（1）每个应用程序只有两个 Activity：MainActivity 和一个可被其他应用程序调用的 Activiy。

（2）每个 Activity 只能被实例化一次。在实验中，我们在系统里指定了三类应用程序，即 Victim（受害应用程序）、Video（良性播放器）和 Mallory（恶意应用程序），创建这三类 App 来模拟场景，能够涵盖网络钓鱼攻击、欺骗攻击、勒索 App 这三类典型的任务劫持攻击案例。

4.4.1 实验环境

本章实验是在 Microsoft Windows10(x64)上完成，处理器为 Intel(R)Core(TM) i5-1035G1，内存为 8GB，硬盘容量为 500GB，使用的程序语言为 JAVA 与 Kotlin，使用 Android Studio 进行开发，虚拟机版本为 Android 9.0 及以上，与真实 Android 设备或虚拟 Android 设备进行交互的工具为 ADB（Android Debug Bridge）。

4.4.2 网络钓鱼攻击

网络钓鱼攻击是任务劫持攻击的一种，攻击者通常将自己的界面伪装成真实的界面，从而获取用户在虚假界面中输入的隐私信息。例如，返回（Back）按钮允许用户在 Activity 的历史记录中返回上一个 Activity，攻击者通过劫持 Back 按钮，误导用户，从而进行网络钓鱼。如使用银行类应用程序时，当用户想要从第三方应用程序 Activity 中返回银行登录界面时，攻击者通过劫持 Back 按钮，使用户在点击 Back 按钮后返回到恶意应用仿

冒银行登录页的界面，此时用户并不知情，并将自己的用户名、密码等隐私信息填入界面，攻击者会立即获取这些信息。

本节设计了两种网络钓鱼攻击方法，图 4.2 展示了这两种方法的状态转换模型。

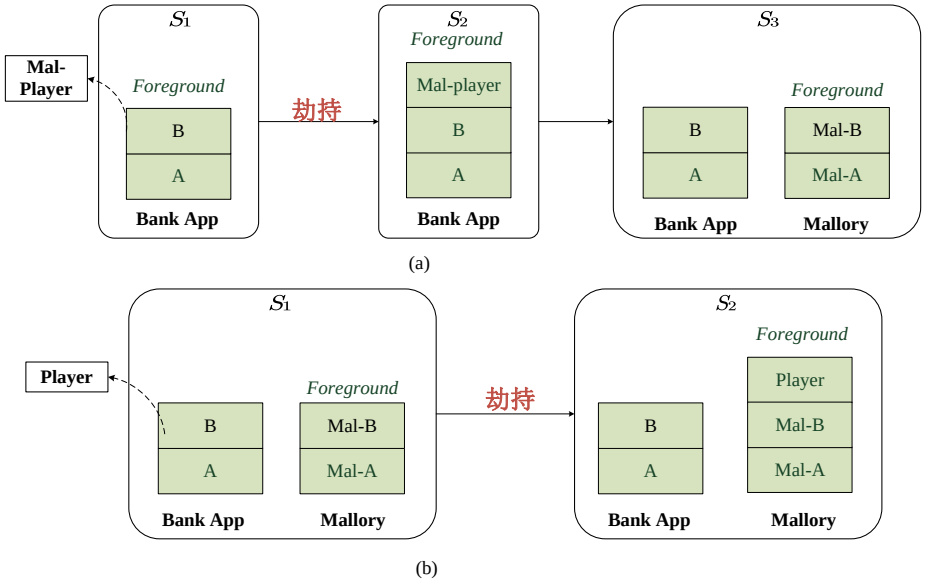


图 4.2 两种网络钓鱼攻击的任务状态转换模型

图 4.2（a）为第一种方法的状态转换模型。在s_1中，银行应用程序 Bank App 任务包含 Activity A 和 Activity B，其中 Activity B 是银行登录界面。任务劫持攻击发生在用户从登录界面点击该 Bank App 的教程视频的过程中（$s_1 \rightarrow s_2$），此时，攻击者将恶意视频播放器 Mal-player 置于 Bank App 任务栈的栈顶，用户从系统弹窗中选择了 Mal-player，因此教程视频由 Mal-player 播放，用户观看完视频后，按下 Back 按钮触发状态$s_2 \rightarrow s_3$的转换，此时，攻击者通过在 Mal-player 活动中重写 onBackPressed()方法来修改 Back 点击事件，并在 Bank App 中创建一个新的恶意任务将其置于任务栈的栈顶，而不是恢复 Activity B。因此，攻击者劫持了 Activity A 和 Activity B，显示为 Mal-A 和 Mal-B 这两个钓鱼 Activity。在这类攻击方法中，恶意应用程序需要伪装成常用的应用程序（例如一个视频播放器），用户才可能会使用。

图 4.2（b）为第二种方法的状态转换模型。在s_1中，有两个钓鱼 Activity 的恶意任务潜伏在后台。任务劫持发生在$s_1 \rightarrow s_2$，当用户启动良性视频播放器 Player 时，该播放器并未加入到 Bank App 的任务栈，而是被置于恶意应用程序 Mallory 任务栈的栈顶，从而在视频播放后用户按下 Back 按钮进入钓鱼 Activity Mal-B。该劫持攻击中良性视频播放器作为新任务启动时，攻击者为其加入一个标志位 FLAG_ACTIVITY_NEW_TASK 或者为其设置 singleTask 启动模式，能够使其置于栈顶，同时 Mallory 将 taskAffinity 恶意设置为良性视频播放器的包名。

这两种攻击方法的不同之处在于用户在第一种攻击中选择了一个恶意的视频播放器，而在第二种攻击中选择了一个良性的播放器。

（1）实验一

我们基于第一种攻击方法设计了网络钓鱼攻击实验一。当用户使用新闻应用程序 Victim 想要观看视频时，由于攻击者通过修改新闻应用的 intent.action，将其设置为恶意播放器的包名，通过代码 4.1 实现，此时，恶意视频播放器 Mallory 置于新闻应用任务栈的栈顶，用户主动选择了 Mallory。然后 Mallory 通过在 Mal-player 活动中重写 onBackPressed()来修改 Back 点击事件，劫持了 Victim 的返回事件，通过代码 4.2 实现。用户使用恶意播放器看完视频后，点击 Back 按钮，并没有返回到原始界面，而是返回到了 Mallory 的劫持界面（Mallory 模仿 Victim 的界面），此时的 Mallory 一直在后台并未退出。仿冒界面中可能诱导用户填写重要信息，导致用户隐私信息的泄露。

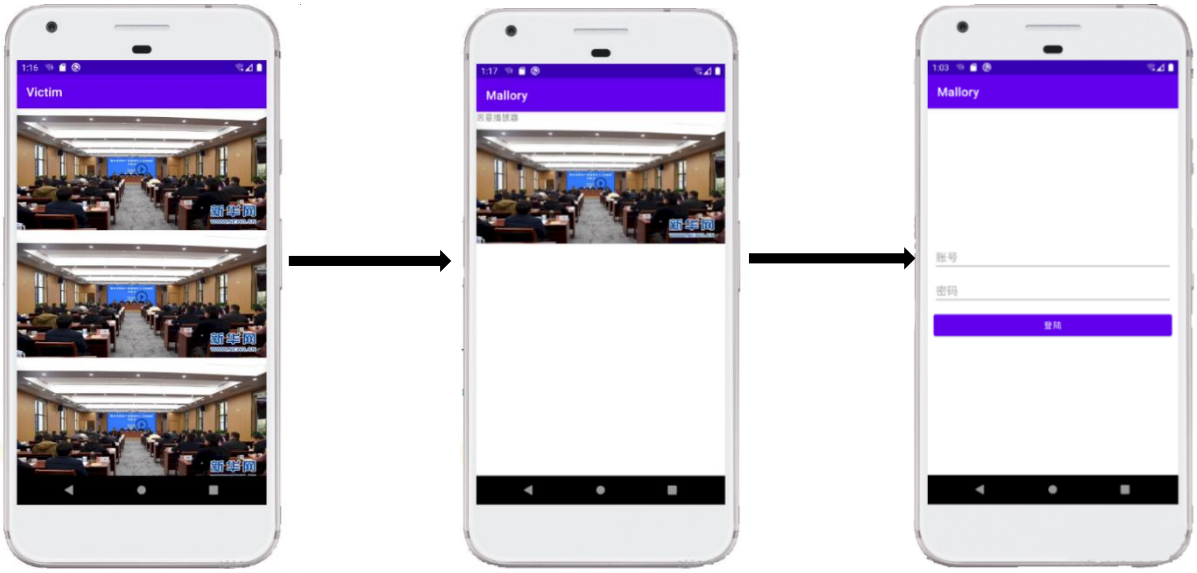
代码 4.1 恶意应用被置于新闻应用任务栈栈顶的关键代码

```
class MainActivity : AppCompatActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    fun toVideo(view: android.view.View) {
        val intent = Intent()
        intent.action = "com.example.mallory.video"
        startActivity(intent)
    }
}
```

代码 4.2 重写 onBackPressed()方法

```
class VideoActivity : AppCompatActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_video)
    }
 override fun onBackPressed() {
    super.onBackPressed()
    val intent = Intent()
    intent.setClass(this,MainActivity::class.java)
    startActivity(intent)
     }
}
```

图 4.3 展示了实验一中钓鱼攻击过程的屏幕截图。图（a）→图（b）：用户启动 Victim 想要观看视频，任意点击一个视频，启动了 Mallory 恶意播放软件；图（b）→图（c）看完视频后点击返回，并没有返回到 Victim 原始界面，而是返回到 Mallory 模仿 Victim 的界面。



（a）　　（b）　　（c）

图 4.3 实验一中钓鱼攻击过程截图

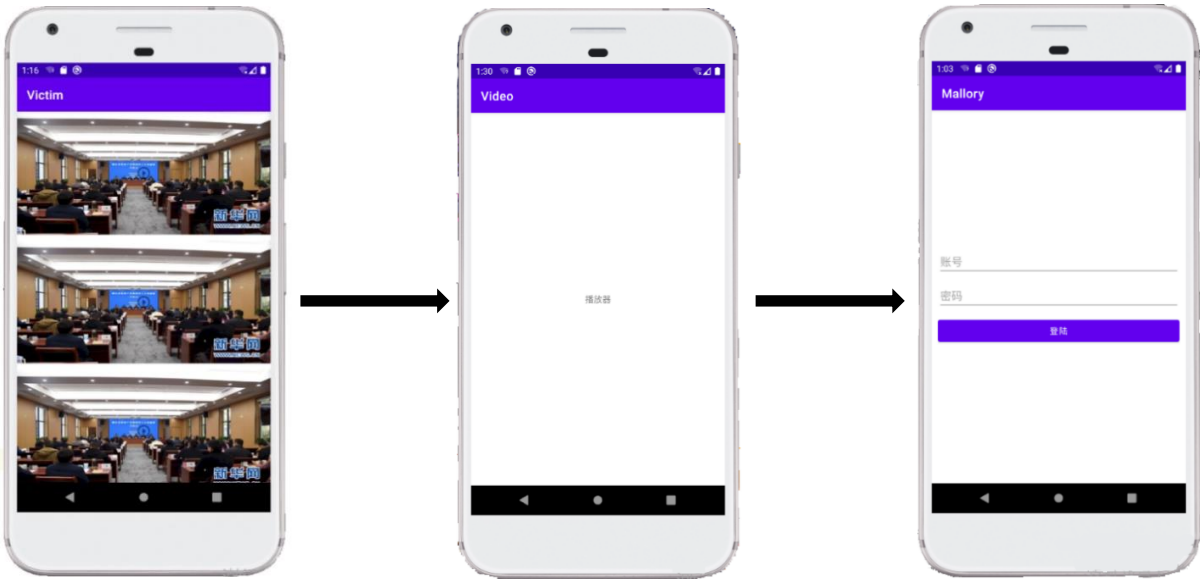
（2）实验二

我们基于第二种攻击方法设计了网络钓鱼攻击实验二。首先当用户启动 Victim 后，任意点击一个视频进行观看，此时启动的是一个良性的播放软件 Video。由于攻击者将 Mallory 中 taskAffinity 设置为 Video 的包名，通过代码 4.3 实现，这时 Mallory 模仿 Victim 界面伪造一个新界面，将新界面置于 Video 的任务栈的栈顶，因此用户返回后会看到该新界面，如果用户在该界面中输入敏感信息，攻击者就会获取这些敏感信息。

代码 4.3 taskAffinity 设置为 Video 的包名

```
1    <activity
2        android:taskAffinity="com.example.video"
3        android:name=".MainActivity"
4        android:exported="true">
5        <intent-filter>
6            <action android:name="android.intent.action.MAIN" />
7            <category android:name="android.intent.category.LAUNCHER" />
8        </intent-filter>
9    </activity>
```

图 4.4 展示了实验二中钓鱼攻击过程的屏幕截图。图（a）→图（b）：用户启动 Victim，点击一个视频进行观看，这时启动了良性播放器 Video；图（b）→图（c）：用户看完视频后返回到 Mallory 的仿冒界面。



（a）　　（b）　　（c）

图 4.4 实验二中钓鱼攻击过程截图

4.4.3 欺骗攻击实验

欺骗攻击是任务劫持攻击的一种，攻击者通过操纵任务劫持条件 taskAffinity、allowTaskReparenting 等启动欺骗界面，窃取用户信息。第 4.2 节中的威胁模型展示了欺骗攻击的状态转换过程。

基于欺骗攻击方法，我们设计了欺骗攻击实验。首先，Mallory 将 taskAffinity 属性值设置为 Victim 的包名，将 allowTaskReparenting 设置为 true，通过代码 4.4 实现。此时，一旦用户启动了 Mallory 并返回主界面后，Mallory 中 MainActivity 进入到生命周期的 Pause 状态，隐藏了之前的 Mallory 界面，用户随后启动 Victim，这时 MainActivity 进入 Resume 状态，劫持界面被展示出来，覆盖了正常的界面，通过代码 4.5 实现。因此用户看到的界面为 Mallory 模仿 Victim 的界面。

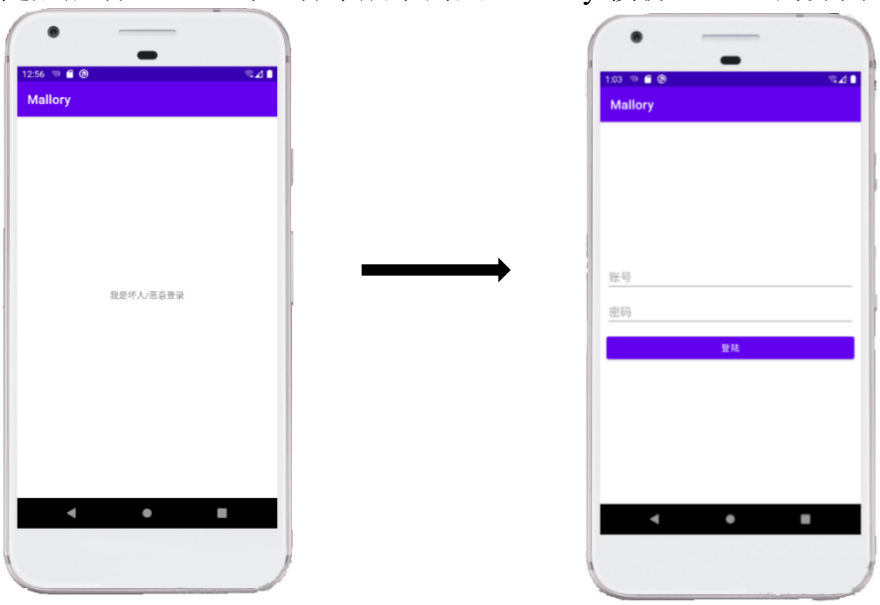
代码 4.4 设置 taskAffinity 和 allowTaskReparenting 属性值

```
1       <activity
2           android:name=".MainActivity"
3           android:taskAffinity="com.example.victim"
4           android:allowTaskReparenting="true"
5           android:exported="true">
6           <intent-filter>
7               <action android:name="android.intent.action.MAIN" />
8               <category android:name="android.intent.category.LAUNCHER" />
9           </intent-filter>
10      </activity>
```

代码 4.5 onPause()和 onResume()方法

```
override fun onPause() {
    super.onPause()
    findViewById<View>(R.id.tv).visibility = View.GONE
}
override fun onResume() {
    super.onResume()
    if (isFirst) {
        isFirst = false
        return
    }
    findViewById<View>(R.id.login).visibility = View.VISIBLE
}
```

图 4.5 展示了欺骗攻击过程的屏幕截图。图（a）→图（b）：用户启动 Mallory 后将其置于后台，随后启动 Victim 时，屏幕展示的是 Mallory 模仿 Victim 的界面。



（a）　　（b）

图 4.5 欺骗攻击截图

4.4.4 勒索 App 实验

勒索 App 是任务劫持攻击的一种，该攻击大致可分为攻击者恶意阻止卸载 App 和攻击者恶意后台收集隐私信息两种类型，其目的都是为了勒索用户。本节基于这两种攻击类型设计了两个攻击实验。

（1）恶意阻止卸载 App 实验

用户卸载应用程序通常有三种方式：（i）从系统设置应用程序卸载；（ii）将应用程序图标拖动到主屏幕上的“垃圾桶”；（iii）在第三方应用程序的帮助下卸载。本节以卸载方式（i）为例，图 4.6 展示了攻击者恶意阻止卸载 App 的状态转换模型。

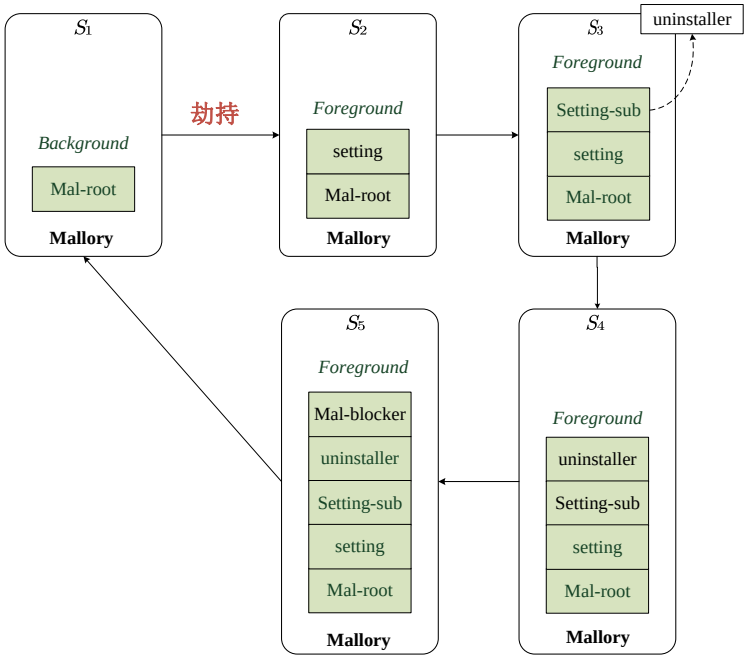


图 4.6 攻击者恶意阻止卸载 App 的状态转换模型

如图 4.6 所示，在s_1中，只有一个来自恶意软件的 Mal-root 在后台等待，由于 taskAffinity 被设置为 Android 系统应用“设置”（Setting）的包名，当用户从主屏幕打开 Setting 时触发任务劫持，将 Setting Activity 置于 Mallory 任务栈的栈顶，状态从s_1转换为s_2。用户通过 Setting 下的子菜单找到应用程序，并点击卸载按钮，显示卸载程序 Activity（Setting-sub）以供用户确认，此时状态从s_2转换到s_3。这时，后台恶意服务会立即启动恶意 Activity（Mal-blocker），并持续监视前台 Activity。Mal-blocker 由新任务启动，与 Setting 具有相同的 taskAffinity 取值，因此被推送到同一任务中，并立即阻止用户卸载程序，这时状态由s_4转换到s_5。由于 Mal-blocker Activity 的 Back 按钮已被恶意程序禁用，因此用户无法访问位于任务栈下方中的 uninstaller，即用户无法点击确认卸载按钮，导致卸载应用程序失败。

基于该攻击方式我们设计了恶意卸载 App 实验。首先，攻击者在 Mallory 中加入定时器，当 Mallory 被用户启动并放入后台时，Mallory 在后台监听用户的操作，通过代码 4.6 实现，间隔 2 秒进行一次检测，检查用户是否启动了 Setting 应用程序，当 Mallory 监听到用户启动了 Setting 应用时，Mallory 会向前台发送弹窗以覆盖设置 Setting 界面，并且 Mallory 屏蔽了 Back 按键，使用户无法进行任何操作，通过代码 4.7 实现。

代码 4.6 Mallory 监听用户操作

```
1Observable.interval(2,TimeUnit.SECONDS).observeOn(AndroidSchedulers.mainThread()).subscribe {
2val activityManager : ActivityManager = getSystemService(ACTIVITY_SERVICE) as ActivityManager
3val componentName = activityManager.getRunningTasks(1)[0].topActivity
4val baseIntent = activityManager.getRunningTasks(1)[0].baseIntent
```

```
5Log.i("MainActivity","$componentName")
6if ("com.android.settings.homepage.SettingsHomepageActivity" == componentName?.className){
7        val intent = Intent()
8        intent.setClass(this@MainActivity,MalBlocker::class.java)
9        startActivity(intent)
10    }
11 }
```

代码 4.7 Mallory 屏蔽 Back 按键

```
1        override fun dispatchKeyEvent(event: KeyEvent?): Boolean {
2            if (event?.keyCode == KeyEvent.KEYCODE_BACK){
3                return true
4            }
5            return super.dispatchKeyEvent(event)
6        }
```

图 4.7 为恶意阻止卸载 App 的实验截图。当用户使用完 Mallory 后将其放入后台，然后通过系统应用程序“设置”卸载 Mallory，点击 “设置”图标时跳出“阻止卸载，给我比特币”字样的勒索信息，致使系统瘫痪用户无法返回。

图 4.7 恶意阻止卸载 App 实验截图

（2）恶意后台收集隐私信息实验

我们设计了恶意后台收集隐私信息实验。通过设置 Intent 属性能够实现界面与界面之间的跳转，该属性中存放了界面跳转之间的各种数据与参数。Mallory 添加了定时器与监听事件，用户启动 Mallory 并将其置于后台，此时，Mallory 通过对任务栈的栈顶 Activity 的 Intent 进行监听，一直在后台收集用户当前界面信息以及 Activity 相关的启动信息等多种信息，并将收集的信息上传至服务器。代码 4.8 为 Mallory 对 Intent 监听并后台收集信息的关键代码。

代码 4.8 Mallory 对 Intent 监听并后台收集信息

```
1 Observable.interval(2, TimeUnit.SECONDS).observeOn(AndroidSchedulers.mainThread()).subscribe
2 (new Consumer<Long>() {
3   ……
4   public void accept(Long aLong) throws Exception {
5   ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
6   ComponentName componentName = activityManager.getRunningTasks(1).get(0).topActivity;
7      Intent baseIntent = activityManager.getRunningTasks(1).get(0).baseIntent;
8      Log.i("MainActivity","敏感信息" + componentName + ":" + baseIntent);
9   }
10}
```

图 4.8 所示为 Android 虚拟机中 Mallory 在后台收集用户敏感信息的证据截图。图中 Mallory 收集了应用程序 Activity 相关启动信息、用户敏感信息等多种信息。例如：

（1）com.example.malloryI/MainActivity:敏感信息 ComponentInfo{com.example.mallory/com.example.mallory.MainActivity}:Intent{act=android.intent.action.MAINcat=[android.intent.category.LAUNCHER]flg=0x10000000cmp=com.example.mallory/.MainActivity}；

（2）system_process I/ActivityTaskManager:STARTu0{act=android.intent.action.MAINcat=[android.intent.category.LAUNCHER]flg=0x10000000cmp=com.example.mallory/.MainActivity} from uid 2000.

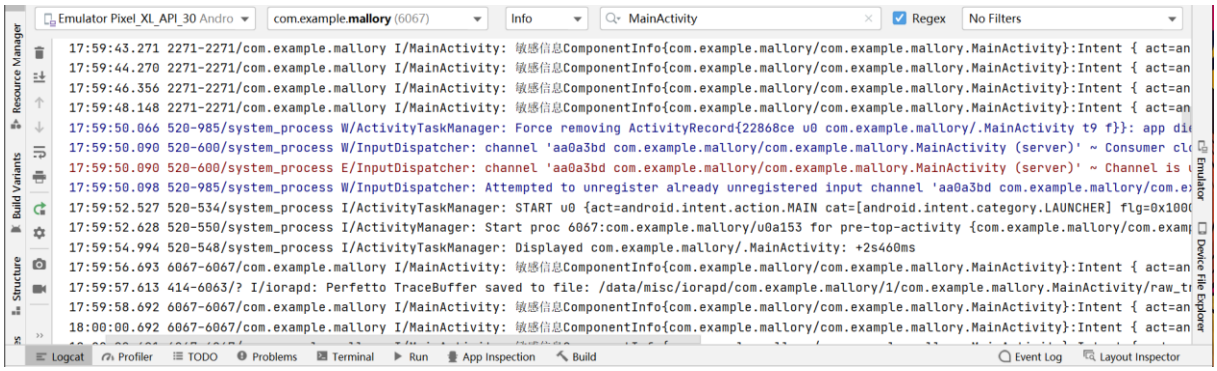


图 4.8 后台收集信息截图

4.4.5 任务劫持条件的现实使用

通过以上实验可得，由于 Android 多任务机制的灵活性，任务状态转换的复杂性，使应用程序易受任务劫持攻击。我们需要一组安全指南预防这类攻击的发生，为此，首先要了解任务劫持条件在实际生活中的使用情况。本节分析了各大应用厂商中五类应用程序：银行类、广告类、短信类、游戏类和新闻类，共 85 个应用程序。对应用程序包进行反编译，对反编译后的代码进行分析，统计得出各个应用程序中任务劫持条件的使用率。

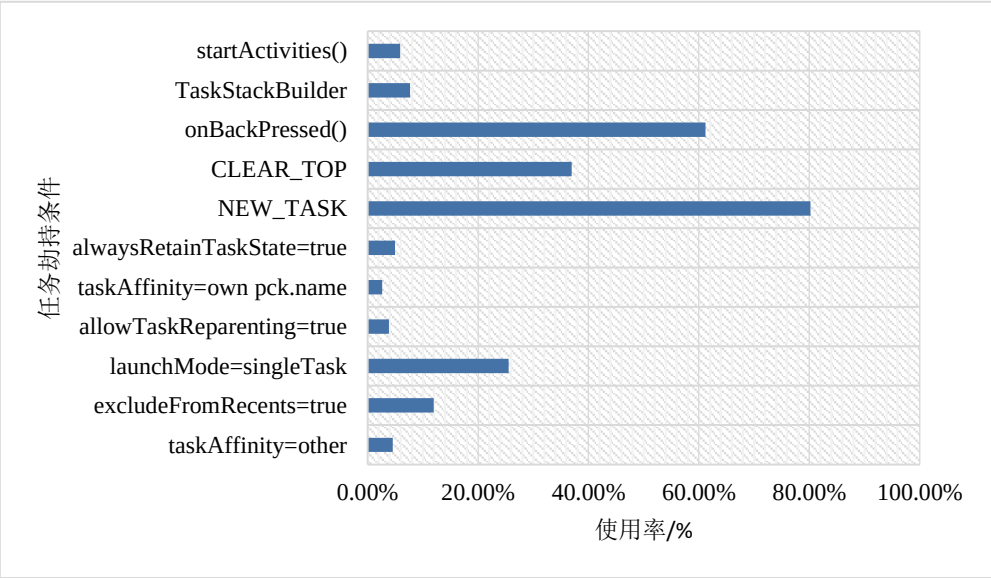


图 4.9 85 个应用程序中使用任务劫持条件的比例

图 4.9 展示了 85 个应用程序中使用每个任务劫持条件的比例。如图所示，我们研究的应用中，7.18%的应用程序显式声明了 taskAffinity，其中，4.55%的应用程序设置它们的 Activity 的 taskAffinity=true，不是他们的包名，此时，如果存在任务冲突，这 4.55%的应用程序可能会干扰彼此的多任务行为，如果一个应用的 taskAffinity 被故意设置为其他应用的包名，它们可能会对其他应用实施恶意行为。在我们研究的应用程序中，FLAG_ACTIVITY_NEW_TASK Intent 标志的使用率为 80.21%，其在大部分应用程序中被用来控制新建 Activity 与任务的关联。大量应用程序采用了 Back 按钮，其中一个原因是 61.19%的应用程序大量使用 onBackPressed()回调函数，这使得应用程序会在 Activity 被销毁之前进行数据清理。3.85%的应用程序设置它们 Activity 的 allowTaskReparenting=true，此时允许当前组件被转移至其他应用组件的任务栈，如果其他应用为恶意应用，那么该应用程序可能被恶意应用实施任务劫持攻击。

4.5 缓解任务劫持的安全指南

通过分析以上实验可得，给定一组能够平衡应用程序安全性和功能性的指南是必要的。因此，我们基于之前的研究得出以下五条安全指南。

（1）在代码中的应用程序包名后，备注开发人员设置的 taskAffinity 属性值。

（2）对 taskAffinity 配置属性的取值进行约束。任何一个 taskAffinity 的取值不应包含其他应用程序的包名，除非这两个应用程序来自同一个开发人员。

（3）当一个应用程序将 taskAffinity 设置为 false（ 默认值），或该应用程序与其他应用程序设置的 taskAffinity 属性值相同时，系统不会无条件将该应用程序的 Activity 置于其他应用程序的任务栈中。

（4）在应用程序中限制 onBackPressed()方法的使用，防止攻击者通过在恶意 Activity 中重写 onBackPressed()方法以修改 Back 点击事件，来劫持应用程序的返回事件。

（5）当一个应用程序的 allowTaskReparenting 设置为 true，且 taskAffinity 的取值为其他应用程序的包名时，系统不会无条件允许当前 Activity 从该应用程序的任务栈移至 taskAffinity 指定的其他应用程序的任务栈中。

4.6 本章小结

本章分析了在 Android 多任务机制下可能产生的任务劫持攻击，实现了五组任务劫持攻击实例，其中包括网络钓鱼攻击（2 组实例），欺骗攻击（1 组实例），勒索 App（2 组实例），再现了由于配置引起的几种劫持攻击过程，例证了配置对漏洞发现所产生的关键作用。最后通过分析攻击实验，统计各大应用厂商中 85 个应用程序使用任务劫持条件的比例，给出了五条缓解任务劫持攻击的安全指南。

5 安卓 App 配置安全性检查方法研究

基于第 3、4 章若干配置规则以及任务劫持攻击相关的配置漏洞研究，本章提出了一种 Android App 配置安全性检查方法，并设计和开发了一个 Android App 配置安全检测工具。使用 CICMalDroid 2020 公开数据集和各大应用厂商中下载的良性应用程序进行实验，验证了工具的有效性。并且该工具对 MobSF 工具中安全配置检测进行了有益的扩展和补充。

5.1 问题分析

对于某些配置而言，即便使用了正确的配置方法也可能会引发安全问题，带来安全隐患。因此，本章设计并实现了一个 Android App 配置安全性检查方法—Android App 配置安全检测工具。

5.2 Android App 配置安全检测工具

Android App 配置安全检测工具工作过程如图 5.1 所示，分为三个模块。

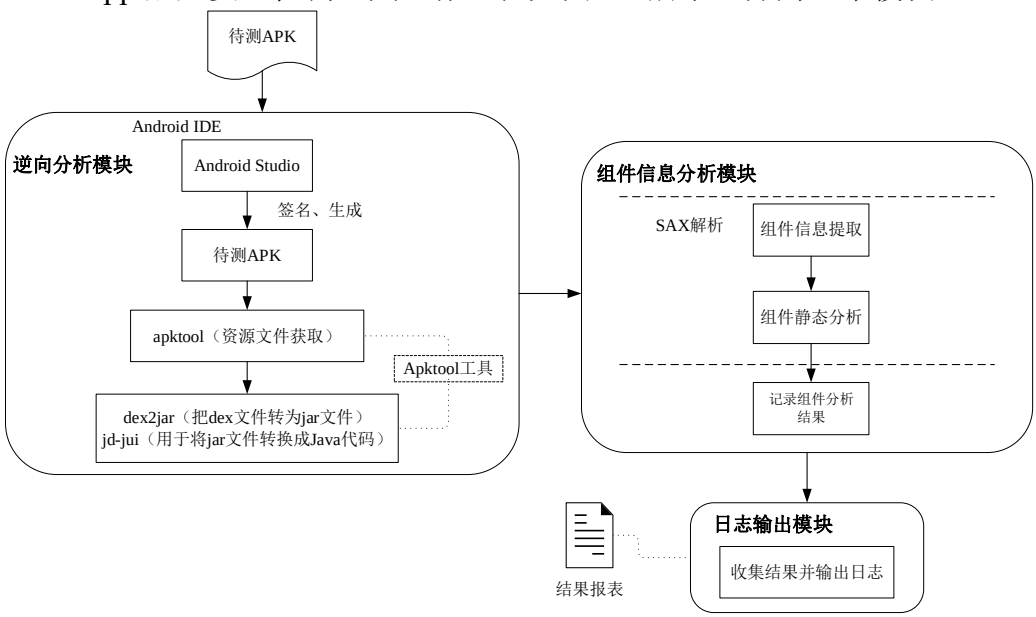


图 5.1 Android App 配置安全检测工具

（1）逆向分析模块：首先遍历 APK 目录下的安装包，判断是否为.apk 文件，并创建以.apk 文件前缀命名的文件对象；然后解压 APK 并将其反编译。APK 文件是 Android 操作系统的可执行文件，由于 APK 是打包好的安装文件，并不能直接进行读取，因此首先要对 APK 文件进行反编译。本章使用 ApkTool 将 APK 文件进行反编译，APKTool 是 Google 提供的 APK 反编译工具，可以最大限度的还原 APK 文件的内容。使用 ApkTool

对 APK 文件进行反编译的命令如下所示：

java -jar apktool.jar d sourse.apk -o output -s -f

命令中的参数 d 表示要对 Apk 文件进行反编译，参数-o 指定了反编译结果的输出目录。

（2）组件信息分析模块：使用基于 SAX 的 XML 分析器提取反编译后的组件详细代码信息，然后解析 Manifest.xml 文件。SAX 是 Java XML 处理的应用接口（Java API for XML Processing），它采用基于事件驱动的处理模式，将 XML 文件转化为一系列事件，由单独的时间处理器来决定如何处理这些事件。SAX 克服了 DOM 的缺点，能够立即对 XML 文件进行分析，而不需要将所有数据处理完成后再对其分析。由于应用程序只在读取数据时对其进行检查，因此不需要将数据存储在内存中，能够提高处理大型文件的效率。

信息提取的内容包括：App 入口类；App 包含的所有 Activity；App 包含的敏感权限；组件的 exported 属性；组件的 launchMode 启动模式；组件的 allowTaskReparenting 属性；组件的 taskAffinity 属性。本章中涉及的敏感权限为 Android 6.0 中定义的危险权限，具体信息如表 5.1 所示。

表 5.1 Android 6.0 危险权限

权限组	危险权限	说明
CONTACTS	WRITE_CONTACTS GET_ACCOUNTS READ_CONTACTS	与此设备上的联系人和配置文件相关的运行时权限
PHONE	READ_CALL_LOG READ_PHONE_STATE CALL_PHONE WRITE_CALL_LOG USE_SIP PROCESS_OUTGOING_CALLS ADD_VOICEMAIL	与电话功能相关的权限
CALENDAR	READ_CALENDAR WRITE_CALENDAR	与用户日历相关的运行时权限
CAMERA	CAMERA	与访问摄像头或从设备捕捉图像/视频相关权限
SENSORS	BODY_SENSORS	与访问人体或环境传感器相关的权限
LOCATION	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION	允许访问设备位置的权限
STORAGE	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE	与共享外部存储相关的运行时权限
MICROPHONE	RECORD_AUDIO	与从设备访问麦克风音频相关权限

SMS	READ_SMS RECEIVE_WAP_PUSH RECEIVE_SMS RECEIVE_SMS SEND_SMS READ_CELL_BROADCASTS	与用户短信相关的运行时权限

（3）日志输出模块：收集并分析测试结果，输出结果日志。通过分析组件配置的取值，判断该组件是否有安全风险，若存在安全漏洞，则输出结果日志，具体判断内容如表 5.2 所示。

表 5.2 App 配置取值及说明

组件配置	取值	说明
入口类	一个	配置正确
	两个及两个以上	配置不当，可能出现多个图标（入口类）共用一个包名，此时只要卸载一个应用程序，与这个 APK 包名相同的程序也都会在桌面上消失，并且从其他应用跳转到该 APK 时，无法区分打开的是哪一个应用程序
敏感权限	危险权限列表中的权限	有安全风险
	非危险权限列表中的权限	无安全风险
exported	false	不允许其他应用组件调用当前组件，因此无安全风险
	true	允许其他应用组件调用当前组件，存在组件暴露的安全风险
launchMode	null	为标准模式，无安全风险
	singleTask	设置 singleTask 可能出现 Activity 被放入新任务栈中的情况，如果新任务栈存在于恶意应用中，那么该应用程序存在网络钓鱼攻击的安全风险
allowTaskReparenting	false	该组件必须在当前的任务栈中不能移动，因此无安全风险
	true	允许当前组件被转移至其他应用组件的任务栈，假定其他应用为恶意应用，则视为该应用程序存在欺骗攻击的安全风险
taskAffinity	默认值 （当前应用的包名）	该组件在默认任务栈中，无安全风险
	其他应用的包名	当前组件可被其他应用启动，假定其他应用为恶意应用，则视为该应用程序存在被恶意应用任务劫持攻击的安全风险

获取应用程序中敏感权限信息以及判断权限存在的安全风险，关键代码为代码 5.1 中 1-22 行。获取组件配置取值信息以及判断这些取值存在的安全风险，关键代码为代码 5.1 中 23-50 行。

代码 5.1 获取信息以及判断安全风险

```
1    private static final String CONTACTS =
2    "android.permission.READ_CONTACTS;android.permission.WRITE_CONTACTS";
3    private static final String PHONE =
4    "android.permission.READ_CALL_LOG;android.permission.WRITE_CALL_LOG;
5     android.permission.READ_PHONE_STATE;android.permission.CALL_PHONE";
6    ……
7    private static HashSet<String> risk = new HashSet<>();
8    public static void output(ParseManifest manifest) {
9       System.out.println("packageName:" + manifest.appPackage);
10      System.out.println("permissions(" + manifest.permissions.size() + "):");
11         for (int i = 0; i < manifest.permissions.size(); i++) {
12            System.out.println(manifest.permissions.get(i));
13            if (CONTACTS.contains(manifest.permissions.get(i))){
14               risk.add("存在联系人泄露风险");
15            }
16            if (LOCATION.contains(manifest.permissions.get(i))){
17               risk.add("存在位置泄露风险");
18            }
19            if (STORAGE.contains(manifest.permissions.get(i))){
20               risk.add("存在存储内容泄露风险");
21            }
22            ……
23      System.out.println("activities(" + manifest.activities.size() + "):");
24      for (int i = 0; i < manifest.activities.size(); i++) {
25         System.out.println(manifest.activities.get(i) + "\tlaunchMode:" +
26 (manifest.launchModes.get(i).isEmpty() ? "default" : manifest.launchModes.get(i)) +
27"\tprocess:" + (manifest.processList.get(i).isEmpty()? "default":manifest.processList.get(i)) +
28     "\tallowTaskReparenting:" + (manifest.allowTaskReparentings.get(i).isEmpty() ?
29     "default" :manifest.allowTaskReparentings.get(i)));
30            }
31      for (int i = 0;i<manifest.allowTaskReparentings.size();i++){
32         if (manifest.allowTaskReparentings.get(i) != null
33         && !manifest.allowTaskReparentings.get(i).isEmpty()){
34              risk.add("存在欺骗攻击风险");
35               }
36            }
37      for (int i = 0;i<manifest.launchModes.size();i++){
38         if (manifest.launchModes.get(i) != null&& !manifest.launchModes.get(i).isEmpty()){
39              risk.add("存在网络钓鱼劫持风险");
40                 }
41            }
42      for (int i = 0;i<manifest.taskAffinitys.size();i++){
43         if (manifest.taskAffinitys.get(i) != null && !manifest.taskAffinitys.get(i).isEmpty()){
44              risk.add("存在任务劫持攻击风险");
45         }
46      }
47         if (exporteds.size()>0){
48              risk.add("存在组件暴露风险");
49         }
50         ……
```

5.3 实验结果分析

5.3.1 实验环境与数据集

本章实验在操作系统 Windows 10(x64)上完成，处理器为 Intel(R)Core(TM) i5-1035G1，内存为 8GB，硬盘容量为 500GB，使用的程序语言为 JAVA，使用 Android Studio 进行开发。

实验中使用的良性 App 来自 CICMalDroid 2020 公开数据集中抽取的 1286 个良性 App（广告类和银行类）以及从各大应用厂商中下载的 518 个良性 App，共有财务、社交、动漫、音乐、广告类、工具、体育等 22 个种类，良性应用程序数据集详细构成如表 5.3 所示。

表 5.3 良性应用程序数据集

良性 App 种类	数量	良性 App 种类	数量
广告类(advertising)	925	银行类(Banking)	361
小部件(wallpaper)	20	社交(social)	14
财务(finance)	6	商务(business)	12
动漫(comic)	12	摄影(photograph)	56
个性化(personalization)	14	生活时尚(lifestyle)	12
工具(tools)	46	体育(sport)	27
购物(shopping)	20	通讯(communication)	36
健身与健康(health&fitness)	32	图书与工具书(book)	30
交通运输(transportation)	19	医务(medical)	20
教育(education)	50	游戏(game)	15
媒体与视频(media&video)	42	音乐与音频(music&audio)	35

5.3.2 工具验证与结果分析

本章针对配置漏洞实现了 Android App 配置安全检测工具，使用数据集对该工具进行验证。本节使用一个 App 为例来进行说明，以下内容为实验结果展示，证明了 Android App 配置安全检测工具的有效性。

该工具打印出包名为“com.mapswithme.maps.pro”的地图类 App 中的所有敏感权限，共 11 个，权限信息如表 5.4 所示。例如，该 App 中包含与 SMS 相关的敏感权限 SEND_SMS、READ_SMS 和 WRITE_SMS，因此其具有泄露用户短信信息的风险；该 App 中包含与 LOCATION 相关的敏感权限 ACCESS_COARSE_LOCATION 和 ACCESS_FINE_LOCATION，因此其具有泄露用户所在位置信息的风险。

表 5.4 App 中部分权限信息输出结果

1 android.permission.SEND_SMS
2 android.permission.READ_SMS
3 android.permission.WRITE_SMS
4 android.permission. RECORD_AUDIO
5 android.permission.WRITE_EXTERNAL_STORAGE
6 android.permission.READ_PHONE_STATE
7 android.permission. READ_CALL_LOG
8 android.permission. CAMERA
9 android.permission.ACCESS_COARSE_LOCATION
10 android.permission.ACCESS_FINE_LOCATION
11 android.permission.GET_ACCOUNTS

该工具打印出该 App 中所有 Activity 列表和数量、入口类名称、每个 Activity 的 launchMode、allowTaskReparenting、taskAffinity、exported 配置属性的取值，部分配置相关信息如表 5.5 所示。例如，（1）该 App 具有 19 个 Activity，其中入口 Activity 为 com.mapswithme.maps.DownloadResourcesActivity，该 App 只存在一个入口类，则配置正确。（2）该 App 中 com.mapswithme.maps.search.SearchActivity 的 launchMode 值为 3，代表该 Activity 的 launchMode 设置为 singleTask，设置 singleTask 可能出现 Activity 被放入新任务栈中的情况，假定新任务栈存在于恶意应用中，则视为App_2存在网络钓鱼攻击的风险。（3）该 App 中 Activity 的 allowTaskReparenting 均为 default（默认值为 false），表示该 Activity 必须在当前应用的任务栈中不能移动，因此该项不会造成欺骗攻击的风险；（4）该 App 中 Activity 的 taskAffinity 均为 default（该 App 包名），因此该项不会造成恶意应用任务劫持攻击的风险。（5）该 App 中 cplatform.surfdesktop.ui.activity.ShareActivity 的 exported 值为 true，则该 Activity 可被外部访问并调用，存在组件暴露的风险。

表 5.5 App 中配置取值部分输出结果

Activity 名称	launch -Mode	allowTask -Reparenting	taskAffi -nity	exported
com.mapswithme.maps.DownloadResourcesActivity	default	default	default	default
com.mapswithme.maps.MwmActivity	2	default	default	default
com.mapswithme.maps.search.SearchActivity	3	default	default	true
com.mapswithme.maps.settings.SettingsActivity	default	default	default	default
com.mapswithme.maps.editor.EditorActivity	3	default	default	default
com.mapswithme.maps.editor.ProfileActivity	default	default	default	true
com.mapswithme.maps.editor.ReportActivity	default	default	default	default
ru.mail.android.mytarget.ads.MyTargetActivity	default	default	default	default
com.facebook.FacebookActivity	2	default	default	default
cplatform.surfdesktop.ui.activity.ShareActivity	default	default	default	true
net.hockeyapp.android.FeedbackActivity	default	default	default	default
com.mapswithme.maps.editor.OsmAuthActivity	2	default	default	default

net.hockeyapp.android.LoginActivity	default	default	default	default
net.hockeyapp.android.PaintActivity	default	default	default	default

该工具打印出检测到的该 App 中由于配置漏洞导致的安全风险信息如图 5.2 所示。（1）该 App 中包含敏感权限 SEND_SMS、READ_SMS 和 WRITE_SMS，因此其具有泄露用户短信信息的风险；包含敏感权限 ACCESS_COARSE_LOCATION 和 ACCESS_FINE_LOCATION，因此其具有泄露用户所在位置信息的风险；包含敏感权限 RECORD_AUDIO，因此其具有音频泄露风险；包含敏感权限 WRITE_EXTERNAL_STORAGE，因此具有存储内容泄露风险；包含敏感权限 READ_PHONE_STATE、READ_CALL_LOG，因此其具有电话状态泄露风险；包含敏感权限 CAMERA，因此其具有摄像头偷拍风险；包含敏感权限 GET_ACCOUNTS，因此其具有联系人泄露风险。（2）该 App 中组件的 launchMode 设置为 singleTask，因此组件存在网络钓鱼攻击风险。（3）该 App 中组件的 exported 设置为 true，允许其他应用组件调用当前组件，因此组件存在组件暴露的安全风险。

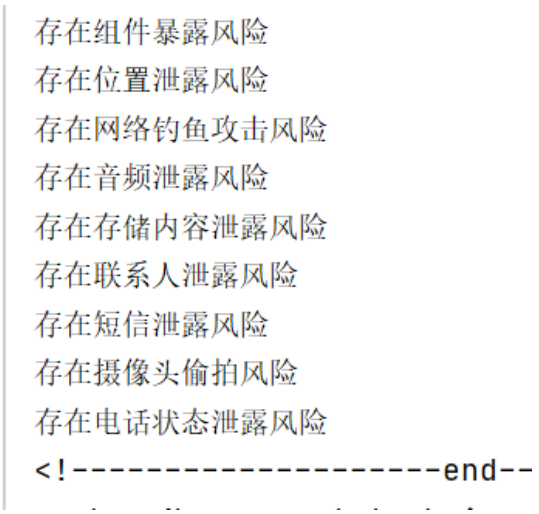


图 5.2 工具输出具有的安全风险实验截图

执行了所有良性应用程序后，我们统计了各个权限组敏感权限的使用频率，并在收集敏感权限时，对所在类别必要的权限不计入总数中，如短信类 App 中的 SMS 相关权限不计入该 App 的敏感权限使用情况中。由图 5.3 可知，在我们研究的应用程序中，有 92.85%的应用程序包含 STORAGE 权限组中的敏感权限，说明大多数良性应用程序存在存储内容泄露的安全风险。有 33.33%的应用程序包含 CALENDAR 权限组中的敏感权限，说明有较少部分良性应用程序存在日历信息泄露的安全风险。而只有 1.26%的应用程序包含 SENSORS 权限组中的敏感权限，说明极少数良性应用程序存在传感器信息泄露的安全风险。

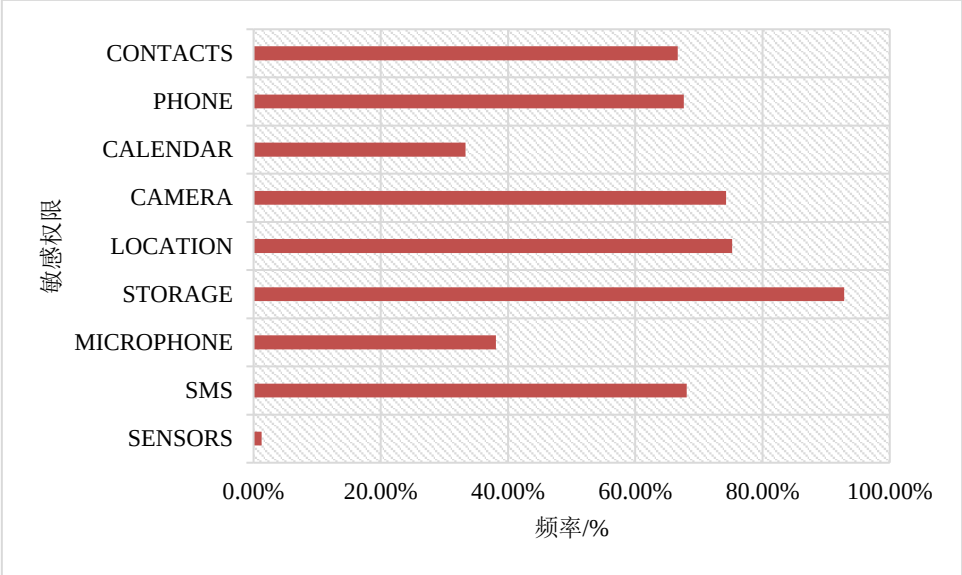


图 5.3 敏感权限使用频率

我们统计了本节研究的组件配置的使用频率。由图 5.4 可知，在本节研究的应用程序中，有 93.80%的良性应用程序的组件配置了 exported=true，说明针对该配置元素给 App 带来的影响，大多数良性应用程序存在组件暴露的安全风险。而只有 6.85%的良性应用程序组件中配置了 allowTaskReparenting=true，说明针对此配置给 App 带来的影响，极少部分良性应用程序存在被恶意应用欺骗攻击的安全风险。

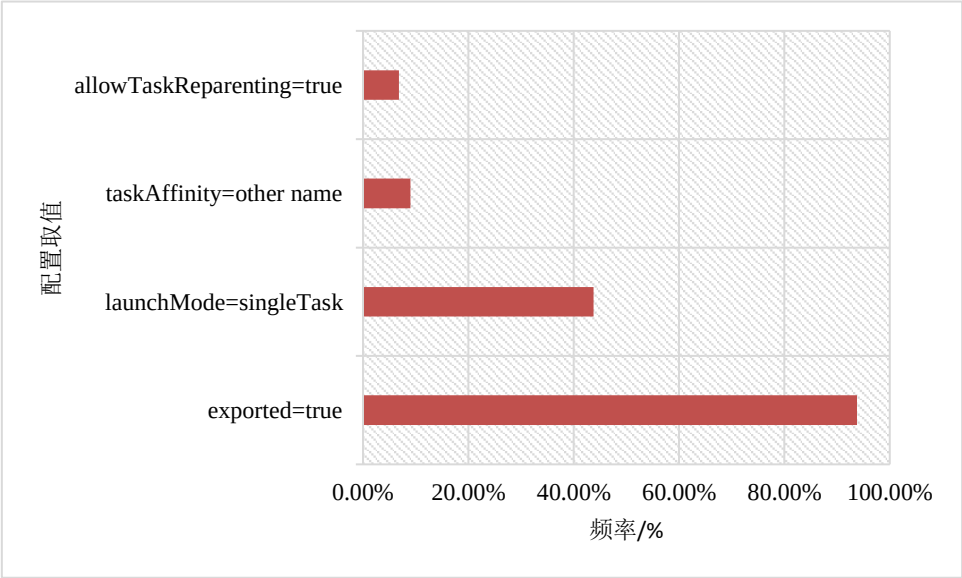


图 5.4 组件配置的使用频率

通过以上实验，验证了 Android App 配置安全检测工具的有效性。

MobSF（Mobile-Security-Framework）是一个开源移动应用程序自动化测试框架，能够对移动应用程序进行静态、动态安全分析。该工具中静态安全分析包括 Manifest 分析、

源码分析和文件分析。MobSF 中 Manifest 安全分析包括的内容如表 5.6 所示。

表 5.6 MobSF 中 Manifest 安全分析

检测内容	安全风险
检查 android:debuggable 是否为 true	如果为 true，那么 App 可以被调试，攻击者可获取调试信息，使该 App 可能泄露关键信息
检查 android:allowBackup 是否为 true	如果为 true，则该 App 可以备份应用数据，存在安全风险
检查 android:testOnly 是否为 true	如果为 true，则在此情况下程序出于测试状态，会暴露程序本身的功能或数据
Activity 启动方式不标准	设置不标准可能导致信息泄露
组件暴露	任何组件都不应设置为可被外部访问，否则存在安全风险
不适当的 Content Provider 权限	Content Provider 如果设置权限为设备上所有 App 均可访问，则有可能导致其中包含的敏感信息泄露
检查 android:scheme 中是否存在 android_secret_code	如果存在该项，则该 App 可能导致加密内容泄漏
二进制短信端口处于监听状态	如果对二进制短信没有进行适当处理，假定所有接收到的短信都来自不可信任源，则应用程序存在安全风险

本章设计并实现的 Android App 配置安全检测工具对 AndroidManifest.xml 中部分配置信息进行了分析和研究，其中前四条为此工具对 MobSF 的有益扩展和补充。

（1）获取 App 入口类，判断应用程序是否包含两个及两个以上入口类。例如，当有两个及以上两个入口类（App 图标）的应用程序共用一个包名时，只要卸载一个应用程序，与这个 APK 包名相同的程序也都会在桌面上消失，并且从其他应用跳转到该 APK 时，无法区分打开的是哪一个应用程序。这种情况可能引发安全问题，导致隐私信息泄露等。

（2）获取组件的 launchMode 启动模式，判断该组件定义的 launchMode 是否为 SingleTask，若是，则可能出现组件被放入新任务栈中的情况，如果新任务栈存在于恶意应用中，那么该应用程序存在网络钓鱼攻击的安全风险。

（3）获取组件的 allowTaskReparenting 属性值，判断组件中 allowTaskReparenting 属性值是否为 true，若是，则允许当前组件被转移至其他应用组件的任务栈中，若其他应用为恶意应用，则该应用程序存在欺骗攻击的安全风险。

（4）获取组件的 taskAffinity 属性值，判断组件中 taskAffinity 属性值是否为其他应用程序包名，若是，则假定其他应用为恶意应用，视为该 App 存在被恶意软件任务劫持攻击的风险。

（5）获取声明的权限信息，判断是否包含敏感权限，若权限信息中包含了敏感权限，则可能造成隐私泄露。

（6）获取组件的 exported 属性值，判断 exported 属性值是否设置为 true，若是，则允许其他应用组件调用当前组件，该 App 存在组件暴露的安全风险。

5.4 本章小结

本章详细的介绍了 Android App 配置安全性检查方法，实现了一个 Android App 配置安全检测工具，详细描述了各个模块的设计及具体实现过程，通过实验证明了该工具的有效性，并对权威工具 MobSF 中的安全配置检测进行了有益的扩展和补充。

6 总结与展望

6.1 工作总结

随着 Android 系统的日益发展，搭载 Android 系统的智能移动终端已经成为主流，每天都有大量的应用程序被上传至应用市场供用户下载，但随即带来的 Android 应用程序安全问题也越来越突出。本文通过分析 Android 官方文档中对组件配置元素的描述存在不确定性、二义性的问题，展开对 Android App 组件配置元素的语义以及各元素之间关系的认定和发掘，并研究了由于配置漏洞所产生的安全问题。本文主要工作总结如下:

（1）为了有效减少开发人员在设计应用程序的过程中产生组件配置漏洞，本文首先针对 Android 官方文档中部分组件配置描述语义模糊的问题，用模糊测试方法和组合测试方法构造测试用例，其中模糊测试方法构造测试用例的策略为基于生成和基于变异两者结合的策略。然后通过对用例运行结果的分析，明确这些配置项的含义，在此基础上进一步采用一阶谓词逻辑描述这些配置项的语义，形成 32 条形式化配置规则，以供设计及开发人员使用。

（2）为了展示由于配置方面的漏洞所引发的应用程序安全问题，本文以三种典型的任务劫持攻击为例，研究了配置方面的脆弱性。首先通过分析任务劫持状态转换模型，详细研究了任务劫持状态转换过程，然后基于模型实现了五组攻击实例，其中包括网络钓鱼攻击（2 组实例），欺骗攻击（1 组实例），勒索 App（2 组实例），再现了由于配置引起的五种劫持攻击过程，例证了配置对漏洞发现所产生的关键作用，最后通过分析攻击实验，统计各大应用厂商中常见的五类应用程序使用任务劫持条件的比例，给出了五条缓解此类攻击的安全指南。

（3）通过对上述配置规则以及劫持攻击相关的配置漏洞的研究，本文提出了一种 Android App 配置安全性检查方法，并设计和开发了一个 Android App 配置安全检测工具，其中包括逆向分析模块、组件信息分析模块、日志输出模块三个模块。最后对该工具做了分析与验证，使用 CICMalDroid 2020 公开数据集和各大应用厂商中下载的应用程序进行了实验，该工具可以分析这些 App 中的六种配置项可能带来的 App 安全问题，我们统计了这些 App 中敏感权限使用频率和 exported、launchMode、allowTaskReparenting、taskAffinity 四种配置项特定取值的使用频率，证明了该工具的有效性。进一步将该工具与现有成熟工具 MobSF 中的配置安全性检测进行了对比，结果表明该工具的检测工作能够形成对 MobSF 的有益扩展和补充。

6.2 工作展望

本文在对 Android App 组件配置元素的语义以及各元素之间关系的认定和发掘、配置漏洞研究等方面取得了一定进展，但还存在一些问题需要进一步深入研究。在未来的工作中，将从如下几个方面开展深入研究：

（1）在研究配置规则的过程中，本文使用模糊测试和组合测试方法进行测试用例的构造。组合测试中，测试用例的生成是一个比较耗时的过程，可以考虑将该方法和云计算技术结合，通过并行执行测试用例的方法来加快测试速度。模糊测试中，变异策略和变异操作，对于提高变异后的输入质量有着非常重要的影响。在现有工作中，由于缺乏对变异结果的分析，变异策略和变异操作有限，产生变异输入的效率不高。我们将在未来研究主流的机器学习技术，合理地利用执行过程中的反馈，学习制定更好的变异策略和变异操作，对于提高新输入的质量，缩短执行时间有着积极的作用。通过以上方法可以更高效的研究更多 Android 组件配置，进而得出更多准确的配置规则。

（2）本文提出的 Android App 配置安全性检查方法虽然能够对 Android 应用程序中的部分安全相关配置进行检测，但是该方法只分析了 AndroidManifest.xml 文件中的部分配置项，后面将进一步对更多种类的配置进行研究。

致 谢

“流光容易把人抛，红了樱桃，绿了芭蕉”，时光稍纵即逝，转眼间，我三年的研究生生活就要结束了，有许许多多的不舍，也有许许多多的感谢。

首先衷心感谢我的导师***老师！您的言传身教、悉心指导与谆谆教诲使我终身受益。您对知识孜孜不倦的追求、对自己学科领域研究的热爱时刻影响着我的思想、行为，让我懂得了如何做事，如何学习。您在和我们的交流中，显现了您的开朗、宽容、豁达，教会了我如何面对生活和学习。从您帮助我们学习探索知识的过程中，您对学术方面的严谨态度时刻提醒着我应当静下心来努力学习。您传授给我的专业知识是我不断成长的源泉，也是完成本论文的基础。千言万语化为一句“谢谢您，感谢遇见您”。

接下来我要感谢我的父母！他们是我做好一切事情的基石，他们是我最坚强的后盾。父母总是最疼爱我们的，每当我因为学习上的事日夜焦虑、崩溃时，给父母打一个电话，他们的鼓励、支持、理解是可以暖心窝的，是我遇到困难时坚持下来的动力源。感恩父母，你们辛苦了！

最后感谢**师姐、***、***师弟、我的室友们、16 号楼宿管阿姨等诸多在生活和学习上给过我帮助和温暖的人，你们见证了我的成长，共同分享我的喜悦，倾听我的烦恼，并帮助我走出困境。谢谢你们在我任何需要帮助的时候给予我最暖心的帮助与鼓励，祝福你们都有一个更美好的明天。

往事历历在目，要感谢的人实在太多，思绪万千，又岂是一个谢字能够表达。唯有在今后的工作和生活中，不断提高自我，严格要求自己，以不负众望。

参考文献

[1] 洞见研报. 2021 年中国手机操作系统行业[EB/OL]. https://baijiahao.baidu.com/s?id=1701330565823407684&wfr=spider&for=pc. 2021-06-01.

[2] 傅建明, 李鹏伟, 易乔, 黄诗勇. Android组件间通信的安全缺陷静态检测方法[J]. 华中科技大学学报(自然科学版), 2013, 41(S2): 259-264.

[3] 韩继登. 基于Android系统组件劫持的漏洞分析[D]. 北京: 北京邮电大学, 2015.

[4] Xiao J, Chen S, He Q, Feng Z, Xue X. An Android application risk evaluation framework based on minimum permission set identification[J]. Journal of Systems and Software, 2020, 163: 110533.

[5] 李秀, 陆南. 基于数据挖掘的Android恶意应用检测方法的研究[J]. 计算机与数字工程, 2019, 47(12): 3089-3094.

[6] 刘倩, 韩斌. Android平台下的基于应用分类和敏感权限挖掘的恶意应用检测方法研究[J]. 计算机与数字工程, 2019, 47(06): 1446-1451+1481.

[7] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification[C]//Proceedings of the 16th ACM conference on Computer and communications security. 2009: 235-245.

[8] 卜同同, 曹天杰. 基于权限的Android应用风险评估方法[J]. 计算机应用, 2019, 39(01): 131-135.

[9] 金俊杰. 基于系统函数拦截的 Android 应用权限细粒度控制方案研究与实现[D]. 南京: 南京邮电大学, 2018.

[10] Meng Z, Xiong Y, Huang W, Qin L, Jin X, Yan H. AppScalpel: Combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in Android applications[J]. Neurocomputing, 2019, 341: 10-25.

[11] Bohluli Z, Shahriari H R. Detecting privacy leaks in android apps using inter-component information flow control analysis[C]//2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC). IEEE, 2018: 1-6.

[12] 李智, 陈金威, 陈世喆, 张金龙. 基于静态污点分析法的 Android 信息泄露研究[J]. 电子质量, 2015(10): 71-74.

[13] Enck W, Octeau D, McDaniel P D, Chaudhuri S. A study of android application security[C]//Proceedings of the 20th USENIX security symposium. 2011, 21-44.

[14] Choi K, Ko M, Chang B M. A Practical Intent Fuzzing Tool for Robustness of Inter-

Component Communication in Android Apps[J]. KSII Transactions on Internet and Information Systems (TIIS), 2018, 12(9): 4248-4270.

[15] 肖卫, 张源, 杨珉. 安卓应用软件中 Intent 数据验证漏洞的检测方法[J]. 小型微型计算机系统, 2017, 38(04): 813-819.

[16] 王国珍, 杨红丽. Android 应用中 Exported Activity 测试途径研究[J]. 计算机系统应用, 2018, 27(09): 262-267.

[17] Fuzzer[EB/OL]. https://blackarch.org/fuzzer.html. 2022-01-26.

[18] American Fuzzy Lop[EB/OL]. https://www.likecs.com/show-203683235.html. 2021-04-01.

[19] Li Y, Chen B, Chandramohan M, Lin S, Liu Y, Tiu A. Steelix: program-state based binary fuzzing[C]//Proceedings of the 2017 Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 627-637.

[20] Rawat S, Jain V, Kumar A, Cojocar L, Giuffrida C, Bos H. VUzzer: Application-aware Evolutionary Fuzzing[C]//NDSS. 2017, 17: 1-14.

[21] She D, Pei K, Epstein D, Yang J, Ray B, Jana S. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.

[22] Rustamov F, Kim J, Yun J B. DeepDiver: Diving into abysmal depth of the binary for hunting deeply hidden software vulnerabilities[J]. Future Internet, 2020, 12(4): 74.

[23] Mandl R. Orthogonal Latin squares: an application of experiment design to compiler testing[J]. Communications of the ACM, 1985, 28(10): 1054-1058.

[24] GB/T 38639-2020, 系统与软件工程 软件组合测试方法[S]. 北京: 国家市场监督管理总局, 国家标准化管理委员会, 2020.

[25] 陈翔, 顾庆, 王新平, 陈道蓄. 组合测试研究进展[J]. 计算机科学, 2010, 37(03): 1-5.

[26] Sherwood G. Effective testing of factor combinations[C]//Proc. Third International Conference on Software Testing, Analysis and Review (STAR'94). 1994.

[27] 包晓安, 杨亚娟, 张娜, 林青霞, 俞成海. 基于自适应粒子群优化的组合测试用例生成方法[J]. 计算机科学, 2017, 44(06): 177-181.

[28] Lei Y, Tai K C. In-parameter-order: A test generation strategy for pairwise testing[C]//Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No. 98EX231). IEEE, 1998: 254-261.

[29] 周进. 基于交互关系的组合测试用例生成算法研究[D]. 湖南: 南华大学, 2021.

[30] Cyber security breaches survey 2020[EB/OL]. https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2020/cyber-security-breaches-survey-2020. 2020-03-26.

[31] Felt A P, Wagner D. Phishing on mobile devices[M]. na, 2011.

[32] Aonzo S, Merlo A, Tavella G, Fratantonio Y. Phishing attacks on modern android[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 1788-1801.

[33] Chen Q A, Qian Z, Mao Z M. Peeking into your app without actually seeing it: {UI} state inference and novel android attacks[C]//23rd USENIX Security Symposium (USENIX Security 14). 2014: 1037-1052.

[34] Fratantonio Y, Qian C, Chung S P, Lee W. Cloak and dagger: from two permissions to complete control of the UI feedback loop[C]//2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017: 1041-1057.

[35] Liu Y, Wang J, Xu C, Ma X, Lu J. NavyDroid: an efficient tool of energy inefficiency problem diagnosis for Android applications[J]. Science China Information Sciences, 2018, 61(5): 1-20.

[36] Android permission[EB/OL]. http://developer.android.com/guide/topics/manifest/permission-element.html. Android developers. 2022-03-01.

[37] Manès V J M, Han H S, Han C, Cha S K, Egele M, Schwartz E J, Woo M. The art, science, and engineering of fuzzing: A survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2312-2331.

[38] 宋博宇. 模糊测试与符号执行相结合的漏洞发现技术研究[D]. 哈尔滨: 哈尔滨工业大学, 2017.

[39] Schuckert F, Katt B, Langweg H. Difficult XSS Code Patterns for Static Code Analysis Tools[M]//Computer Security. Springer, Cham, 2019: 123-139.

[40] 倪涛, 张瑞武, 叶星. 基于语料库及语法变异的浏览器 Fuzzing 安全测试[J]. 信息工程大学学报, 2018, 19(03): 369-372+384.

[41] 章晓芳, 冯洋, 刘峘, 陈振宇, 徐宝文. 众包软件测试技术研究进展[J]. 软件学报, 2018, 29(01): 69-88.

[42] 左万娟, 虞砺琨, 王小丽, 黄晨. 航天嵌入式软件测试用例典型设计缺陷研究[J]. 计算机测量与控制, 2019, 27(10): 36-40.

[43] Aghajani E, Bavota G, Linares-Vásquez M, Lanza M. Automated documentation of android apps[J]. IEEE Transactions on Software Engineering, 2019, 47(1): 204-220.

附 录

攻读硕士学位期间学术成果

[1] **, **. Research on Android Application Security Configuration Rules Based on Fuzzing Test[C]//2022 International Symposium on Computer Applications and Information Systems(ISCAIS 2022). (已录用)

攻读硕士学位期间获奖情况

[1]西安科技大学第三届 “研究生机器人创新设计大赛” 三等奖