

2020 届硕士学位论文

分类号: \_\_\_\_\_ 学校代码: 10269

密 级: \_\_\_\_\_ 学 号: \*\*\*\* \* \* \* \*



华东师范大学

East China Normal University

硕士 学位 论文  
MASTER'S DISSERTATION

论文题目:

面向工业界仿冒应用的大规模实证研究

院 系: 计算机科学与技术学院

专业名称: \*\*\*\*\*

研究方向: \*\*\*\*\*

指导教师: \*\*\* \*

学位申请人: \*\*\*

2020 年 3 月

Thesis for master's degree in 2020      University Code:      10269  
                                                Student ID:      \*\*\*\*  
                                                \*\*\*\*

# EAST CHINA NORMAL UNIVERSITY

## A LARGE-SCALE EMPIRICAL STUDY ON INDUSTRIAL FAKE APPS

Department:	School of Computer Science and Technology
Major:	*****
Research Direction:	*****
Supervisor:	***
Candidate:	***

March, 2020

## \*\*\* 硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
***	***	*****	主席
***	***	*****	
***	***	*****	

# 摘要

作为世上最受欢迎的移动操作系统，Android 拥有庞大的应用生态环境，其中包含各式良性应用，也包含恶意应用、重打包应用和仿冒应用。得益于广泛的前人工作，业界对各类恶意、重打包应用有了充分理解，提出了对应检测机制与防护措施；而前期调研结果表示，学术界对仿冒应用进行的研究较为匮乏，该认识匮乏很可能带来隐患。

目前，对仿冒应用的研究面临以下几点问题与挑战：(1) 未有一个公开可用的大规模仿冒应用数据集可供研究者使用。收集数据时需要同时考虑数据规模、来源多样性与代表性，本身就是一个难点；(2) 业界对仿冒应用的概念较为模糊，仿冒应用的基本特征尚未明晰，阻碍了后续的检测工作；(3) 仿冒应用作者的行为特征不明确，无法为业界提供预防或拦截仿冒应用的适当建议。

为获得关于仿冒应用的第一手资料，本文采取实证研究的方式，从以下几方面对仿冒应用进行了深入的分析和研究：

- 收集了体量较大的仿冒应用样本集，改善仿冒应用数据稀缺问题。本研究从多个应用来源入手爬取数据，最终筛选出 5 万个余个仿冒应用，还一并爬取了 36 万余条应用评论数据，可为后续工作如应用特征提取、应用行为分析等提供数据支持。
- 进行首个针对 Android 仿冒应用特征分析进行的大规模研究。基于上述数据集，本研究对仿冒应用进行了多方面解析，提供了仿冒应用的基本特征解读结果，并提供案例分析。分析结果可帮助多个群体抵御仿冒应用威胁。
- 利用元数据对应用开发者进行行为画像。结合应用收集时获得的元数据，本研究从开发者证书、应用发布时间多个角度对仿冒应用开发者开展了一系列行为画像，该画像反映的行为特征可为应用市场提供监管思路。

- 提出了可行的排名欺诈行为检测方法。排名欺诈行为与仿冒应用同属于移动应用黑灰产业。为确认排名欺诈行为与仿冒应用之间的关联，本研究提出了一种针对排名欺诈行为的检测方法。该方法可推广至与评论相关的领域中。

综上，依托收集到的大量数据，本研究对仿冒应用进行了全面的分析，并对多方面受众提出了对应的实用建议，对仿冒应用的认识提升和普通用户的安全意识唤起均有积极影响。本文的研究成果能对相关研究者有一定启发作用，为后续工作提供数据支持和探索方向。

**关键词:** Android 应用程序, 仿冒应用, 实证研究, 数据分析, 排名欺诈

# ABSTRACT

As the most popular mobile operating system in the world, Android owns a huge and complete application ecosystem, in which not only all kinds of benign apps that offer people convenience but also malware with vicious intention and fake apps with unknown purposes are included. Thanks to our full understanding of malware, it is now possible to monitor malware on the industrial level. Preliminary investigation indicates, however, that there is no study in academia aims at fake apps so far. The lack of corresponding knowledge could lead to hidden dangers.

In order to obtain the first-hand data about fake apps, this study adapt the methodology of empirical study and glean data directly from the industry. Considering the problem that existing crawler frameworks are not able to achieve targeted collection, this paper design and implement a fake app filter framework called FakeRevealer whereby the large-scale data collection is done and the fake app data is filed as a dataset. Based on the filtered dataset, this paper interprets the characteristics of fake apps from three different perspectives: fake apps' characteristics, factors affecting fake apps' quantity and the developing trends of fake apps. Additionally, case studies are attached after each perspective, completing the first interpretation on Android fake apps. One step further, in order to verify whether fake apps interact with another chain in the mobile underground industry — ranking fraud, this paper conducts a series the ranking fraud detection using the fake apps' ranking and comment data retrieved from an app market. Aim to the drawbacks of existing detecting methods, this paper proposes two innovative approaches to complete the detection.

In terms of data collection, this study obtains approximately 140,000 data entries leveraging FakeRevealer, where one entry represents an app sample. In terms of characteristic interpretation, this paper provides knowledge like fake apps' naming tendency

and the strategies the fake app developers deploy to evade app markets' regulatory mechanisms. What's more, our case-studies expose the fact that fake app developers counterfeit different apps in different forms and app markets lack cooperative supervision. All the information above does help in understanding fake apps. In terms of comment analysis, manual inspection infers the proposed two innovative detecting approaches are both effective. Meanwhile, this study confirms that fake app developers do utilize ranking fraud to boost their rank.

In summary, this paper completes an innovative study from two aspects, namely fake apps characteristics Interpretation and fake apps' comment analysis. Hopefully, this paper can provide a clear vision toward the fake app and its nature through the data and analysis provided.

**Keywords:** *Android Application, Fake App, Empirical Study, Data Analysis, Ranking Fraud*

# 目 录

<b>摘要</b> . . . . .	i
<b>ABSTRACT</b> . . . . .	iii
<b>目录</b> . . . . .	v
<b>第一章 绪论</b> . . . . .	1
1.1 研究背景 . . . . .	1
1.2 研究现状 . . . . .	3
1.2.1 重打包应用的研究现状 . . . . .	3
1.2.2 恶意应用的研究现状 . . . . .	4
1.2.3 其他类型应用的研究现状 . . . . .	5
1.2.4 针对应用评论的相关研究现状 . . . . .	6
1.2.5 研究现状小结 . . . . .	7
1.3 拟采用的研究方法 . . . . .	7
1.4 本文拟解决的关键问题 . . . . .	8
1.5 本文主要工作 . . . . .	9
1.6 本文组织结构 . . . . .	9
<b>第二章 相关背景介绍</b> . . . . .	11
2.1 Android 相关背景介绍 . . . . .	11
2.1.1 Android 签名证书机制 . . . . .	11
2.1.2 重打包技术 . . . . .	14
2.1.3 Android 系统权限 . . . . .	15
2.2 实证研究相关背景介绍 . . . . .	15
2.2.1 软件工程中的实证研究 . . . . .	16
2.2.2 实证研究方法论 . . . . .	16
2.2.3 有效性验证标准 . . . . .	18

2.3	本章小结 . . . . .	19
<b>第三章</b>	<b>结合实例分析的仿冒应用特征解读 . . . . .</b>	<b>20</b>
3.1	研究概况 . . . . .	20
3.2	仿冒应用的基本特征 . . . . .	20
3.2.1	安全证书与仿冒应用的对应关系 . . . . .	21
3.2.2	仿冒样本与原版应用的相似度 . . . . .	22
3.2.3	案例 1. 持有官方安全证书的可疑样本 . . . . .	25
3.3	影响仿冒应用数量的因素 . . . . .	27
3.3.1	仿冒应用的来源 . . . . .	28
3.3.2	其他因素对仿冒样本数量的影响 . . . . .	29
3.3.3	案例 2. 游戏类别下的仿冒应用 . . . . .	33
3.4	仿冒应用的发展轨迹 . . . . .	35
3.4.1	仿冒应用的研发延迟 . . . . .	35
3.4.2	仿冒应用安全证书的存活时长 . . . . .	37
3.4.3	仿冒应用的行业变迁 . . . . .	39
3.4.4	案例 3. 与大量仿冒应用相关联的仿冒应用安全证书 . . . . .	39
3.5	本章小结 . . . . .	41
<b>第四章</b>	<b>面向仿冒应用的排名欺诈验证方法 . . . . .</b>	<b>42</b>
4.1	研究概况 . . . . .	42
4.2	仿冒评论数据收集 . . . . .	42
4.2.1	评论数据概览 . . . . .	43
4.2.2	基础分析 . . . . .	43
4.3	仿冒应用与排名欺诈关联验证 . . . . .	45
4.3.1	排名欺诈检测初探 . . . . .	45
4.3.2	基于评论用户可信度的排名欺诈排查 . . . . .	46
4.3.3	基于评论内容相似度的排名欺诈排查 . . . . .	49
4.3.4	人工复核 . . . . .	51

4.4	本章小结 . . . . .	53
<b>第五章</b>	<b>总结与展望 . . . . .</b>	<b>54</b>
5.1	总结 . . . . .	54
5.2	展望 . . . . .	55
	参考文献 . . . . .	57
	攻读学位期间发表的学术论文. . . . .	66
	致谢 . . . . .	67

## 插 图

图 1.1 Google Play 应用商店架上应用总数变化趋势 . . . . .	1
图 2.1 数字签名技术的两个步骤 . . . . .	13
图 3.1 对 App 各项属性的统计结果 . . . . .	23
图 3.2 各大应用市场应用详情页 . . . . .	25
图 3.3 从不同应用市场中收集到的应用数量以及各市场仿冒率 . . . . .	28
图 3.4 游戏类 App 及其仿冒样本 . . . . .	33
图 3.5 仿冒延迟总体分布 . . . . .	36
图 3.6 仿冒应用安全证书存活时间分布 . . . . .	38
图 3.7 每季度爬取到的仿冒样本数量（2015 年第四季度到 2018 年第三季度） . . . . .	38
图 4.1 各仿冒应用在 360 手机助手商店中的评论数量与评分分布 . . . . .	43
图 4.2 所有 856 个应用在商店中的评论数量与评分分布 . . . . .	44
图 4.3 两种数据组的应用评论重合率结果 . . . . .	50
图 4.4 应用评论取样 . . . . .	52

## 表 格

表 3.1 安全证书/仿冒应用数量对应表 . . . . .	21
表 3.2 持有官方安全证书的可疑样本 . . . . .	26
表 3.3 目标 App 与其相关统计 . . . . .	31
表 3.4 由“61ed377e85d386a8dfee6b864bd85b0bfaa5af81”签署的部分样本 .	40
表 4.1 各应用用户可信度权重及对应排名（一） . . . . .	48
表 4.2 各应用用户可信度权重及对应排名（二） . . . . .	48
表 4.3 评论重合率结果 . . . . .	50

# 第一章 绪论

## 1.1 研究背景

随着移动市场于近年逐渐兴起，Android 系统作为一个主流的移动端操作系统也在蓬勃发展。数据分析机构 StatCounter 资料显示，Android 市场占有率自发布之日起就在逐年稳步增长。截至 2020 年，Android 系统已经占据全球移动端市场份额的 74.3%<sup>[1]</sup>。与此同时，Android 应用<sup>1</sup>的数量也伴随着 Android 市场的蓬勃发展节节攀高。仅 Android 官方的应用商店 Google Play 就在 2017 年中新上架了近一百万个可供下载的应用程序。虽然因为各种原因，Google Play 上的应用数量在 2018 年有所回落，但如图 1.1 所示，应用市场上目前仍有近三百万个可用的应用程序，Android 应用市场依然充满活力<sup>[2]</sup>。

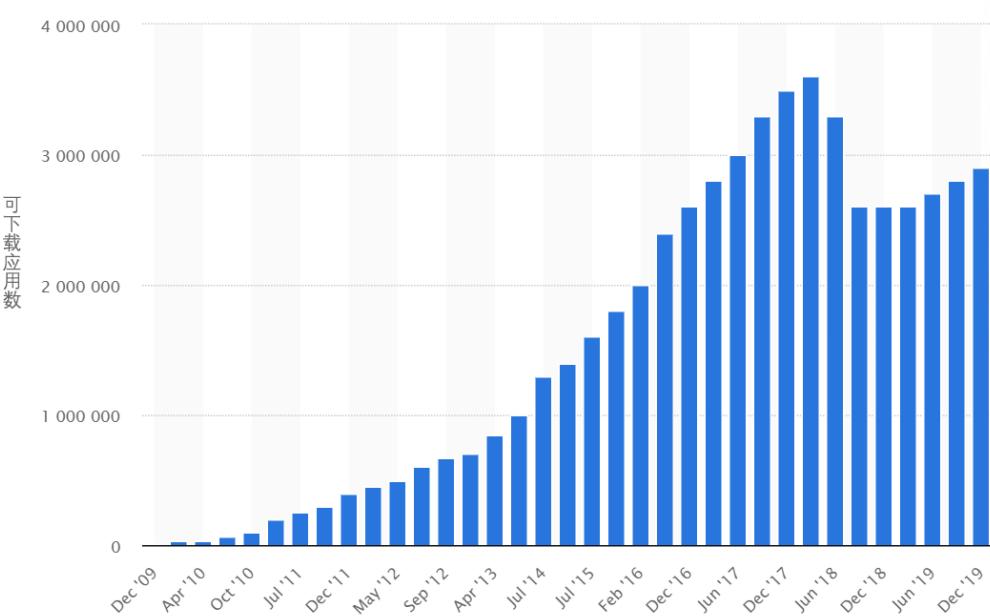


图 1.1: Google Play 应用商店架上应用总数变化趋势

<sup>1</sup>应用、App 在本文中均用以指代 Android 应用程序，不再区分

伴随着应用数量爆发式增长的，还有欣欣向荣的移动黑灰产<sup>1</sup>。一方面，随着开发移动应用的门槛逐渐下降，开发一个移动应用的成本已经远低于开发一个桌面级应用所需的成本<sup>[3]</sup>；另一方面，移动应用功能在实现上的灵活性<sup>[4]</sup>也增加了移动应用的复杂度，让其分析变得更加困难。两方面结合，为黑灰色应用涌入移动端发展提供了绝佳的温床。

黑灰产是以侵害用户、原应用作者或其他第三方利益为手段，凭此或通过其他可疑方式牟利的产业。现阶段已知的移动黑灰产包括恶意应用的编写与传播产业、重打包应用的制作产业和在应用市场上提供批量虚假评价的产业等。

本文提及的仿冒应用指代模仿市面上某热门应用、甚至外观与该热门应用相差无几，但并非由该热门应用开发者发布的移动应用。此类应用的目的是诱导用户下载以赚取流量，甚至是窃取用户信息、触发有害行为从而盈利。故根据上述定义，仿冒应用也是移动黑灰产业链中的一环。根据的前期观察，本文发现仿冒应用以两种不同的形式出现。第一类为模仿应用，这类应用具有和原版热门应用相似的外观（比如名字、图标等），诱导用户下载。而第二类—高仿应用<sup>[5,6]</sup>更采用与原应用一模一样的外观，乃至有相同的版本号，其中部分应用就是直接通过对原应用重打包做成的。

这些仿冒应用不仅大大损害了原应用开发者的利益，也侵犯了用户的权益。可以联想一个普遍场景：当用户尝试通过应用市场搜索安装一个应用，应用市场往往返回多个无论是名字或是图标都十分相像的结果。在未有明确指引的情况下，用户十分有可能安装一个仿冒应用。即使用户最后将仿冒应用删除，并重新装上正版应用，也浪费了时间和人力成本。而前人研究<sup>[7]</sup>显示，应用中的某些恶意行为可被自动触发。万一用户下载的仿冒应用中包含此类恶意行为，将会对用户数据安全造成更大的损害。

综上所述，仿冒应用严重影响用户搜索与下载应用时的安全和体验。为了消除仿冒应用带来的隐患，无论是研究者、应用市场或是普通用户，都应对仿冒应用的特征与行为有所了解。

---

<sup>1</sup>即移动端黑色产业与灰色产业，下文同

## 1.2 研究现状

尽管客观上存在对仿冒应用进行了解的需求，本研究对移动应用领域的前期调研显示，针对仿冒应用进行的实证研究尚缺乏，也未有公开可用的仿冒应用数据集。因此，本节讨论与仿冒应用同属于移动灰黑产的重打包应用和恶意应用，以及其他移动应用领域相关方向的研究现状。

### 1.2.1 重打包应用的研究现状

重打包指恶意开发者对原应用解包、篡改内容之后再将应用重新打包的技术<sup>[8]</sup>。重打包应用侵害了应用原作者的知识产权，甚至可能具有恶意行为，因此也属于移动黑灰产范畴。目前针对重打包应用的相关工作分为三类，分别针对重打包应用的检测和防止应用被重打包，还有关于重打包的实证研究工作。

在重打包应用检测方面，早期工作通过比对两两应用之间的信息获得应用之间的相似度。有基于应用指令序列的方法使用模糊哈希的方法提取出应用的摘要信息，如 DroidMOSS 和 DroidAnalytics<sup>[9,10]</sup>；也有使用静态数据依赖构建程序依赖图 (Program Dependency Graph, PDG) 比对应用的方法，如 DNADroid<sup>[11]</sup> 和 AnDar-win<sup>[12]</sup>，Centroid<sup>[13]</sup> 甚至为应用中的每个函数构建了三维控制流图 (3-Dimensional Control Flow Graph, 3D-CFG)，将三维控制流图聚合后，通过检测不同应用在控制图聚合后的质心位置判断应用间的相似程度；还有算法通过比对抽象语义树检测重打包应用，如 Hanna 等人的 Juxtapapp<sup>[14]</sup> 利用 K-gram 模型对应用的二进制操作码序列进行比对，CLANdroid<sup>[15]</sup> 通过分析五种语义特征点（比如代码中的标识符和调用到的 Android API 等），以检测相似应用。上述算法一来因为需要使用静态分析方法对代码进行解析，在遇到使用过代码混淆处理（如反射和代码加密）的应用时性能往往较差；二来需要基于两两比对进行检测，复杂度较高，在应用规模增大时会导致庞大的计算量。为处理静态分析带来的问题，有学者提出了利用动态分析比对应用的方法，如 Wu 等人使用 HTTP 流量对应用行为进行建模<sup>[16]</sup>。亦有研究使用信息可视化方法检测重打包应用：ViewDroid<sup>[17]</sup> 通过重建和比对应用的视图筛选出重打包应用，DroidEagle<sup>[18]</sup> 则利用 UI 布局相似性检测，Kywe 等人比对应用中的文本与图像相似性以寻找相似应用<sup>[19]</sup>，Soh 等人分析应用运行时收集的 UI 信息

以检测重打包行为<sup>[20]</sup>。而从避免两两比对、减小算法复杂度方面考虑，有学者提出利用第三方库进行检测的办法，如 CodeMatch<sup>[21]</sup> 筛选出应用中使用的第三方库代码后，计算并比对剩余部分代码的哈希值，获取不同应用的相似程度；Wukong<sup>[22]</sup> 也分两步检测重打包应用，但与 CodeMatch 相比，其第二步使用了基于计数的代码克隆检测手段，而非基于哈希的技术。

防止应用被重打包的研究多基于检测原有代码是否被更改而开展。Zhou 等人提出了两个基于改版 Dalvik 虚拟机的对抗工具：其中，AppInk<sup>[23]</sup> 利用一种“水印”技术（Watermarking）对抗重打包，但需要利用一个改版的 Dalvik 虚拟机进行文件检查；另一工具 DIVILAR<sup>[24]</sup> 为应用的 Dalvik 代码提出了一种新编码形式，同时也需要修改用户设备 Android 系统中的 Dalvik 虚拟机以对新的编码进行解码。Luo 等人提出了在代码中注入大量检测节点的方法对抗重打包，并推荐开发者进行代码混淆，以避免令重打包开发者发现注入的检测节点<sup>[6]</sup>。与之相似，Zeng 等人也提出了在代码中注入逻辑炸弹的方法，并在研究中讨论了多种避免重打包开发者发现代码注入的策略<sup>[25]</sup>。

相关实证研究方面，Khanmohammadi 等人利用对重打包应用进行了一次详尽的实证研究<sup>[8]</sup>。该实证研究利用已有数据集 AndroZoo<sup>[26]</sup>，得出了“重打包应用的主要目的为利用广告获得收入”的结论，并回答了包括“何种类型的应用会被重打包”、“重打包对原应用属性更改如何”在内的五个研究问题，增进了研究者对重打包应用的理解。

### 1.2.2 恶意应用的研究现状

针对恶意应用的研究可分为两个方面，分别是面向恶意应用检测的研究与面向恶意应用进行实证研究工作。

恶意应用检测方法可分为几个方向。早期研究通过静态分析手段对恶意应用进行检测，包括 Enck 等人提出的 Kirin，在应用安装时检测应用权限，从而拦截恶意应用的安装<sup>[27]</sup>；Felt 等人提出的 Stowaway<sup>[28]</sup> 利用 API 调用分析和权限分析，寻找权限过大（Overprivileged）的应用；RiskRanker<sup>[29]</sup> 则提出了一个两阶恶意应用检测手段，先筛选出具有 root 行为和可能导致隐私泄露的应用，再找出其中具有

危险 Dalvik 代码加载行为的应用。然而，受静态分析方法本身的限制，此类研究在遇到代码混淆时多被影响性能。因此，有研究者利用动态分析技术对恶意应用进行检测。TaintDroid<sup>[30]</sup> 和 DroidScope<sup>[31]</sup> 均在沙箱中对应用行为进行监视，TaintDroid 利用污点分析技术寻找应用的信息泄露行为，DroidScope 则从 Android 系统的不同层面分析应用的可疑行为。Zhou 等人<sup>[32]</sup> 提出了 DroidRanger，利用应用行为比对与动态加载分析的方式，成功地从 5 个应用市场的 204,040 个应用中找出了 211 个恶意应用。ParanoidAndroid<sup>[33]</sup> 提出了将恶意检测方法迁移到云端的概念，将设备上的所有操作同步到云端服务器，从而在不影响设备本地功能的情况下进行恶意行为检测。无论是静态分析方法或是动态分析方法，都需要人工对恶意行为的模式进行描绘，而用人工手段提取行为模式，不仅对领域知识有较高要求，还往往是费力费时的。针对此问题，研究者开始利用机器学习技术检测恶意应用。Chen 等人从大规模数据集中总结出了恶意应用的两类特征，再将其用于分类器模型训练，提出了检测工具 StormDroid<sup>[34]</sup>，与之类似的方法还有 CrowDroid<sup>[35]</sup>，DroidMat<sup>[36]</sup>，Adagio<sup>[37]</sup>，MAST<sup>[38]</sup>，DroidAPIMiner<sup>[39]</sup> 与 Drebin<sup>[40]</sup>。

在面向恶意应用的实证研究方面，Felt 等人的研究<sup>[41]</sup> 仔细剖析了来自多个不同平台的 46 个恶意程序样本以了解这些样本的激励机制，揭示了这些样本的运行机制和行为策略，为后人抵御此类恶意行为提供参考。Zhou 和 Jiang<sup>[7]</sup> 搜集了来自多个主要恶意应用家族的、超过 1,200 个恶意应用样本，并系统性地描绘了这批样本的不同性质，包括其安装手段、激活机制和其如何执行有效负载（实现恶意行为）。虽然恶意应用与仿冒应用有重合之处，但两者有一定区别，相关理论知识直接套用并不合适。

### 1.2.3 其他类型应用的研究现状

Andow 等人针对灰色应用的研究<sup>[5]</sup> 从 Google Play 应用商店中采集了多个应用样本。该研究将样本分类，定义了高仿应用（Imposter）、移动广告应用（Madware）等 9 种不同的灰色应用。灰色应用指不具有明显恶意行为，但应用意图存疑、又或是会向系统申请过多权限的应用程序。因此，这类应用不是恶意应用，但由于其盈利方式可疑，可将其归入移动黑灰产内。本文的高仿应用定义参照了该文献的内

容。

另一方面, Hu 等人针对市面上的欺诈约会应用进行了一次实证研究<sup>[42]</sup>。该研究中的欺诈约会应用声称自己为约会应用, 实则诱导用户开通会员服务。分析发现, 用户在此类应用中遇到的其他“用户”多数不由真人控制, 而是具有预设对话模板的聊天机器人。此外, 研究还分析了该类应用的商业模式与分发渠道, 评估了受害者的数量, 对此类应用的生态进行了详尽解读。该研究还通过直接检测相同评论和标记可疑用户的方式对此类应用进行排名欺诈检测。比起本研究中提出的方法的, 该研究采用的两种方法均比较原始, 对结果准确程度有一定影响。

Chen 等人发布了漏洞检测工具 Ausera<sup>[43]</sup>, 该工具通过静态分析技术与敏感词识别, 对手机银行应用进行自动化三段式风险评估。同期, 他们在一次实证研究中利用 Ausera 从 693 个手机银行应用中检出超过两千个漏洞, 向对应的 60 家银行发送了漏洞报告, 并检查了新发行的手机银行应用以跟进漏洞修补进程<sup>[44]</sup>。在被检出漏洞的 60 家银行中, 21 家确认了该研究反馈的漏洞, 进行了对应修复, 作者随后根据修复状况进行案例研究, 分别面向银行、学者与政府, 共给出了 9 条用于改善手机银行应用安全性与市场环境的实用建议。与之类似, 本研究也从多个方面出发, 提出了防范仿冒应用、净化市场环境的实用建议。

#### 1.2.4 针对应用评论的相关研究现状

近年来, 除了利用应用程序牟利以外, 移动黑灰产业也开始进入应用市场评论的领域。相关工作也可以分为检测与相关的实证研究两类。

检测方面, Zhu 等人提出了一个利用评价历史记录检测应用是否具有排名欺诈行为的研究<sup>[45]</sup>。该研究提出, 一般应用的排名在上升期 (Raising phrase) 和下降期 (Recession phrase) 之间会有一段维持期 (Maintaining phrase), 维持期过短且排名在短时间内剧烈波动的应用很可能具有排名欺诈行为。然而, 用该方法进行检测需要持续收集应用在市场上的评分数据, 因此对数据采集有一定要求。Lim 等人分析了差评水军 (Review spammers) 的特征并将其用于建模, 以检测类似行为<sup>[46]</sup>; Xie 等人利用图论挖掘评论中开展共谋攻击 (Collusion attack) 的用户群组<sup>[47]</sup>, 并在后续研究中将该方法拓展成一个检测系统<sup>[48]</sup>。与之相似, Chen 等人同样使用了基于

图论的方法检测共谋攻击<sup>[49]</sup>, Hooi 等人的 FRAUDAR 通过寻找二分图中的密集子图寻找关系特别密切的用户与应用, 从而找出可能提供排名欺诈服务的可疑用户与使用了该服务的应用<sup>[50]</sup>。

实证研究方面, Rahman 等人对 Google Play 中虚假评论行为的实证研究<sup>[51]</sup> 在公开确认该产业链存在的同时, 也揭露了该产业的行为模式、生存情况甚至从业人员收入水平等方面的信息。

作为黑灰色产业链下游环节, 虚假应用评论(即排名欺诈行为)已在前述研究中被证明与欺诈约会应用存在关联。因此, 排名欺诈也很可能与仿冒应用相结合, 为移动黑灰产从业者牟取更多利益, 有必要开展相关分析进行确认。

### 1.2.5 研究现状小结

综合上述分析, 与仿冒应用同属黑灰产的重打包应用和恶意应用, 甚至是日常生活中常用的手机银行类应用, 均有对应研究可提供参考。前人的实证研究也收集了较为大量的重打包应用、恶意应用和手机银行应用数据, 并提供了重打包应用与原版差异、恶意应用传播方式与负载执行方式、手机银行应用常见漏洞等指导性信息, 帮助专业人员改善现有移动应用环境。除此以外, 亦有学者对于移动应用相关的应用评论领域进行研究, 且给出了对应领域知识。相比之下, 仿冒应用的相关数据和研究成果都乏善可陈。

此外, 在上述研究中, 无论是对恶意应用、重打包应用或是对可疑评论进行检测, 都需要有一定领域知识或洞见(Insight)作为理论支撑。对仿冒应用而言, 学术界与工业界对其认知均较贫乏, “基于新见解进行拓展研究”式的研究手段尚未适用于仿冒应用。因此, 着眼于领域知识梳理的实证研究更适用于仿冒应用在当下的研究场景。

## 1.3 拟采用的研究方法

上述研究现状说明, 实证研究在多个研究方向上提供了研究数据或领域知识, 有一定实用价值。当下的仿冒应用研究, 既缺乏可用的数据集, 也缺乏对应的基本特征知识。作为基于数据采集和数据分析的技术, 实证研究方法正好可缓解仿冒

应用领域知识和数据集的缺失，是适用于本课题的技术手段。本研究将采用实证研究方法，对仿冒应用进行研究。

实证研究是一种基本研究技术，旨在针对特定方法在实际应用场景下的真实状况或对应产物进行数据收集、调查与分析，以让研究者了解事物的工作原理或使理论得以验证，其中数据收集是十分重要的一部分。前人总结<sup>[52]</sup> 对实证研究的数据收集方法给出了严格指引，指出数据收集时需要确保数据的准确性与全面性。因而，科学地进行数据收集是本研究面临的一个关键问题。

## 1.4 本文拟解决的关键问题

上一节提到，科学地进行数据收集是本研究面临的一个关键问题。此外，本研究还有两个关键问题，一并列举如下：

- 如何科学地获取针对仿冒应用的数据？ 由于仿冒应用数据缺失，本文只能自行收集数据。数据搜集是科研工作中公认的难点，为提供一次全面的研究结果，除了数据需要有一定规模以外，搜集的目标应用也需要具备多样性，此外还必须获得不同应用市场上的数据，增加研究的代表性。为此，如何获得大量真实、有效、全面而有代表性的数据将是本文研究的一个关键点。
- 如何对仿冒应用进行理解？ 仿冒应用虽然普遍存在于各大应用市场中，但尚未有研究对其进行分析，无论是应用市场、研究者还是用户，都缺乏对仿冒应用的理解。为能较全面地认识仿冒应用，需要从不同角度对其进行解读。因此，从应用基本属性、应用作者行为等多角度对仿冒应用进行解析，是本文研究的重点之一。
- 仿冒应用的市场反馈如何？ 想要较为全面地了解仿冒应用的生态，仅从基本特性分析是不够的。为拓展对仿冒应用的认识，本文选择从市场反馈角度入手，与其他应用对比，分析仿冒应用是否有排名欺诈行为。这是本文研究的另一重点，也是本文的创新点。

## 1.5 本文主要工作

为解决上述三个关键问题，本研究完成了如下主要工作：

- **全面的数据集整理** 借助仿冒应用收集框架 FakeRevealer，本研究尽可能全面地收集了来自 29 个应用来源的约 14 万个应用样本，并从中筛选出 5 万个仿冒样本，整理成已知的首个仿冒应用数据集，并进一步从 360 手机助手商店中搜集了来自近 27 万名用户的 36 万条数据，以供其后的排名欺诈检测使用。
- **仿冒应用数据画像与现实案例分析** 本研究从基本信息与开发者行为两个方向入手，对仿冒引用进行了较为全面的数据画像。然后，本文还从数据集中挑选出了三组数据，进行了三个案例分析，深化在数据画像中得到的领域知识。
- **提供实用建议** 基于在实证研究中的发现，本文分析了现有应用市场的不足之处，并分别面向普通用户、应用市场和相关从业者提供了实用建议。
- **针对仿冒应用的排名欺诈检测** 本研究先利用前文提及的 FRAUDAR 进行排名欺诈行为检测，但该方法存在一定局限性，未能提供较为良好的结果。因此，本研究提出两个排名欺诈检测方法，分别从评论用户可信度与评论内容相似度出发，以验证仿冒应用与排名欺诈之间是否存在关联，完善对仿冒应用的生态了解。

## 1.6 本文组织结构

本文共分为五章，环绕着本研究的数据搜集、分析过程和分析结果展开，各章节内容如下：

**第一章** 主要提供了本文的研究背景、相关工作、本文拟采用的研究方法、拟解决的问题与本文的主要工作。

**第二章** 从 Android 应用的构建流程出发，介绍了 Android 安全证书机制、应用市场与黑灰产业的关系和移动黑灰产简介。

**第二章** 根据研究现状提出了本研究中关注的研究问题，阐述了本研究中数据收集的关注点并对采集到的数据进行概述。

**第三章** 先从实验设置与数据搜集出发，介绍仿冒应用收集框架 FakeRevealer 与收集到的数据，再从多个视角分类提出并解说针对这批仿冒应用数据得到的发现，并从数据中挑选三个具有代表性的案例进行分析，以案例进一步深化分析结果。

**第四章** 针对仿冒应用进行了排名欺诈检测。针对现有检测方法的不足，本文先后提出两个具有创新性的方法对排名欺诈行为进行排查，并将排查结果与前一章的仿冒应用进行比对。

**第五章** 对本文工作进行总结，并对下一步工作进行展望。

## 第二章 相关背景介绍

本章对本研究涉及的相关知识作背景介绍，包括与研究主体相关的 Android 背景知识介绍以及与研究方法相关的实证研究相关知识介绍。

### 2.1 Android 相关背景介绍

本研究在收集、筛选仿冒应用样本时，主要使用了应用证书进行筛查，因此先在本节对 Android 的签名证书机制作介绍。此外，鉴于本研究在实证研究中讨论了仿冒应用的权限、仿冒应用与重打包应用的联系，本节也将相关知识作为 Android 背景知识一并介绍。

#### 2.1.1 Android 签名证书机制

##### 1) 加密散列函数与消息摘要技术

消息摘要（Message Digest）又称 hash 值或 hash，是将数据输入加密散列函数（Cryptographic hash function）后所得的输出结果。对消息采用加密散列函数得到消息摘要的行为被称为提取摘要或者提取 hash 值。加密散列函数是散列函数的一种，可理解为一类加密算法，有如下特性<sup>[53]</sup>：

- 对任何给定的信息，都能算出一个 hash 值 无论输入的信息长度如何，加密散列函数均能在多项式时间内给出一个具有固定长度的 hash 值。该长度具体由加密散列函数采取的加密算法决定，如被广泛使用的 MD5 算法返回结果的长度为 128bit（可被表示为 32 位十六进制数字），SHA-1 算法返回长度为 160bit（40 位十六进制数字）等。一般来说，可认为返回的结果越长，采用的加密算法越安全。
- 难以由函数的输出值反推出函数的输入值 加密散列函数是单向函数，即易于凭借一定输入获得一个输出，但难以利用该输出反推出原本的输入。基于

此特性，想要通过一个 hash 知道原本的数据几乎是不可能的。理论上，基于后文描述的无碰撞特性，可采用暴力爆破的方法，对所有可能的输入计算 hash，再比对所得结果，找出某个 hash 对应的原输入。然而，由于前文提及的特性，加密散列函数的输入空间是无穷大的，在一般情况下要通过暴力破解法反推 hash 的原输入几乎不可能。

- 理想的加密散列函数具有确定性与无碰撞特性 对于理想的加密散列函数，任何输入均只对应唯一输出，即不会发生结果碰撞（不同输入得到相同输出）的情形，且对同样的输入也总会得到同样的输出。受制于现实条件，目前的加密散列函数都有可能发生碰撞。然而，好的加密散列函数应该只有极低的概率会发生碰撞事件。基于此特性，可使用通过较安全的加密散列函数处理的 hash 进行安全性验证。
- 对输入的细微改变会导致输出变得截然不同 该特性被称为雪崩效应，是密码学上的概念，具体指由输入值产生细微变化（如一个 bit 的翻转）导致的输出的不可区分性改变（即输出中每个二进制位有 50% 几率发生翻转）<sup>[54]</sup>。基于这种特性，加密散列函数的结果充满随机性，使得攻击者难以推测利用函数输出反推函数输入。

基于以上特点，由各类加密散列函数处理得到的信息摘要具有一定安全性，常用于数据完整性检查、数字证书签名和数据校验等场景中。

## 2) Android 中的数字签名

数字签名技术是一种用电子信息对数据进行签名，从而保证数据完整性与可信度的技术，是不对称加密算法与消息摘要技术结合的一种应用场景。不对称加密算法中，信息发布者有两把密钥，分别是公开的公钥和不能公开的私钥。由于公钥加密的信息仅可由私钥解密，私钥加密的信息也仅可用公钥验证，信息发布者可利用不对称加密算法确保发布的数据不被篡改。然而，非对称加密算法在加解密过程中效率并不高，当信息中心的数据量较大时会产生性能问题。因此，数字签名技术还结合了消息摘要技术确保性能。

具体而言，数字签名技术在实际应用中可分为两个步骤（见图 2.1），分别为消息发送方的签署与消息接收方的验证。在签署步骤中，发送方先使用加密散列函

数处理待发送的数据，得到待发送数据的 hash 值  $H_A$ 。其后，发送方使用自己的私钥对该  $H_A$  进行加密，此处称加密的结果为签名。签名随后将与一份证书（其中附有发送方的公钥和前述的加密散列函数算法信息）一起附于数据中，组成图 2.1 中经过数字签名的内容。而验证步骤中，接收方在收到上述经过数字签名的内容后的处理可分为两部分：一部分为取出其中的数据，根据证书中提供的加密散列函数处理该数据，得到 hash 值  $H_B$ ；另一部分为取出签名与证书中的公钥信息，用公钥对签名进行解密，即可得到原数据的 hash 值  $H_A$ 。只要  $H_A$  和  $H_B$  相同，就能确认数据在传递过程中没有被损坏或篡改。

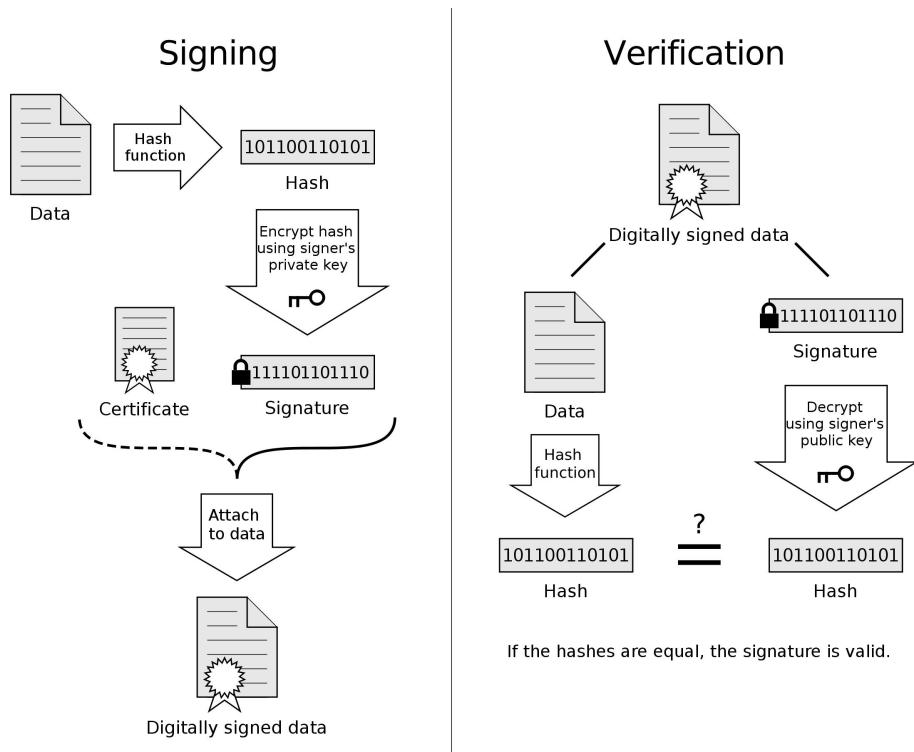


图 2.1: 数字签名技术的两个步骤

在 Android 系统中，数据签名技术的重要应用之一是对应用安装文件（APK 文件，其本质为一个压缩文件）的验证，APK 文件构建尾声和 Android 系统对 APK 文件的安装前验证两个节点分别对应数字签名技术中的签署和验证步骤。

在 APK 文件构建步骤尾声，APK 打包器开始如下签名过程，并生成三个文件：(1) 遍历 APK 中所有文件，对每个文件都提取摘要（常用 SHA-1 或 SHA-256 算法），并将每个文件的路径和 hash 值分别编成条目，把所有条目写入一个新文件

*MANIFEST.MF* 中；(2) 使用同样的加密散列函数对 *MANIFEST.MF* 提取摘要，写入第二个新文件 *CERT.SF* 中，再对 *MANIFEST.MF* 中的每个条目分别再进行一次摘要提取（二次摘要），每个摘要结果连同条目对应的文件路径也写入 *CERT.SF*；(3) 对 *CERT.SF* 的内容使用 APK 发布者的私钥加密，加密结果为一个数字签名，将该签名、签名流程使用的加密散列函数算法以及包含公钥信息的数字证书一同写入第三个新文件 *CERT.RSA* 中。三个新文件与 APK 中原有的内容组成了经过数字签名的内容。

Android 系统对 APK 文件的安装前验证则是与上述流程近似的过程：(1) 扫描 APK 中的所有文件，按 *CERT.RSA* 提供的加密散列函数算法对每个文件提取摘要，和 *MANIFEST.MF* 中的每个条目比对；(2) 用 *CERT.RSA* 提供的公钥对签名进行解密，将解密得到的内容与 *CERT.SF* 的内容进行比对；(3) 对 *MANIFEST.MF* 每个条目进行二次摘要，将二次摘要与 *CERT.SF* 对应条目中的摘要进行比对。原理上，若 APK 中文件被第三方篡改，被篡改文件的摘要就与 *MANIFEST.MF* 中的对应摘要不符；若进一步篡改 *MANIFEST.MF* 的对应摘要，则会导致其二次摘要与 *CERT.SF* 中的对应内容不符；篡改方没有 APK 发布者的私钥信息，无法产生新的签名将 *CERT.RSA* 中的原有签名替换，即使其再篡改 *CERT.SF* 中的二次摘要，也无法令上述第(2)步中的解密结果与被篡改后的 *CERT.SF* 内容相吻合。因此，验证流程中，任何一处产生不相符均会导致验证失败，从而令 Android 系统终止安装流程。

### 2.1.2 重打包技术

上述的数字签名技术可保证原 APK 被篡改后可被发现，但并不能防止对 APK 的篡改行为。篡改方对 APK 进行修改之后，重新对修改后的內容进行构建的行为被称为重打包。重打包主要采用了反编译的技术手段对原 APK 进行篡改：(1) 利用 Apktool<sup>[55]</sup>, dex2jar<sup>[56]</sup>, jd-gui<sup>[57]</sup> 等开源工具拆解 APK 包，反编译出其中包括 java 源码在内的內容；(2) 对反编译得到的內容进行修改，包括增删代码，替换资源文件等操作；(3) 对修改后的內容重新进行打包、签名（使用篡改方的私钥进行签名），完成构建。经过重打包的应用 APK 文件可通过 Android 系统对 APK 的安

装前验证，进而安装至用户设备中。然而，由于重打包应用使用了篡改方的密钥进行重新签名，APK 的 *CERT.RSA* 中文件包含的公钥也是篡改方的公钥，而非原开发者的公钥信息。按此原理，可直接使用 APK 包内 *CERT.RSA* 包含的公钥信息辨认开发者，分辨出正版应用与仿冒应用。

### 2.1.3 Android 系统权限

权限机制是 Android 系统用以保证 Android 应用安全性的系列机制之一，Android 系统对大量 API 均设置了对应权限，应用未获得权限时调用相关 API 会产生错误。此机制被用于实现最小权限原则<sup>[58]</sup>，应用一开始只被赋予最基本的权限，以防止应用得到未获许可的数据，从而导致滥用行为。Android 系统定义的权限中，有相当一部分与用户隐私信息相关，如读取联系人列表（REDA\_CONTACTS）、读取 SD 卡内容（READ\_EXTERNAL\_STORAGE）等权限。因此，可通过一个应用申请的权限对该应用进行数据画像，评估该应用对用户信息的把握程度与可能造成的风险，亦可以借助前人研究提供的工具 PScout<sup>[59]</sup> 分析应用是否存在滥用权限（申请了权限但不进行相关操作）的可疑行为。

开发者要使应用获得权限，需预先将对应权限在 APK 中的 *Manifest.xml* 中声明。在 Android 6.0 版本（API 23）推出之前，系统安装应用时将提示用户该应用可获得的所有权限，应用在安装后可任意使用 *Manifest.xml* 中声明的权限。为了保护用户隐私，Android 6.0 版本对权限有一定改动，应用常见权限如获取时区、读取 WiFi 状态等，可在申请后直接使用，但当应用操作涉及到敏感权限时，系统将弹窗请求用户批准。

## 2.2 实证研究相关背景介绍

实证研究是本研究主要采用的研究方法，因此本节将介绍实证研究的相关背景与常用方法。

### 2.2.1 软件工程中的实证研究

在软件工程领域，由于理论研究与工程实践结合较为紧密，学者采用需要检验理论在实践中的落地情况（如程序语言提供的特性是否被良好地理解与使用<sup>[60]</sup>、是否在工程实践中体现优势等<sup>[61]</sup>问题）或是调研现实应用场景中产生的现象（如恶意应用、重打包应用等在移动端市场出现的实际问题<sup>[5,7,41,62]</sup>），实证研究技术是十分适合的方法。因此，有越来越多的研究者使用实证研究技术对研究主体进行探索<sup>[5,7,8,41–44,60–68]</sup>，并为相关研究领域贡献了宝贵的领域知识。随着实证研究技术在软件工程领域中的认可度逐渐提高，软件工程顶级会议 FSE 与 ICSE 近年分别新设立了面向工业界的投稿分区，鼓励学者多基于应用场景进行实证研究，探明业界现状。

理论方面，为向软件工程领域进行实证研究的学者提供方法论支持，Kitchenham 等人基于自身在评阅软件工程项目的经验和一份针对医学研究者的研究指引，总结出了一份对于软件工程实证研究的初步准则（Preliminary guideline）<sup>[52]</sup>，对实证研究的数据收集方法、实验设置方法等各个步骤都给出了严格指引，如在数据收集时需要确保数据的准确性与全面性，需要保证实验的可重复性等。Seaman 则结合实例，提供定性分析的方法建议<sup>[69]</sup>。在实证研究可采取的具体方式方面，East-erbrook 等人提出了面向软件工程领域的实证研究方法建议<sup>[70]</sup>，为软件工程实证研究提供方法论参考，文中将实证研究方法论分为受控实验、案例分析、调查研究、社会学意义研究与混合方法途径等多类，以适用于不同场景。

### 2.2.2 实证研究方法论

上一小节提及，软件工程的实证研究方法有前人文献可供参考<sup>[70]</sup>。本小节将对介绍文献中总结可用的实证研究方法。

#### 1) 受控实验（Controlled Experiments）

受控实验是用于验证假设的一种研究方法。研究者通过对实验中的自变量（Independent variable）进行小心控制，以测量自变量对一个或多个因变量（Dependent variable）产生的影响。受控实验可使研究者精确地确认各个变量之间如何关联，并特别适用于探究各变量间是否存在因果关系。为使结果具有说服力，受控实验中

除自变量以外的其他变量不应对实验结果产生影响。

## 2) 案例分析 (Case studies)

案例分析是软件工程领域最常用的实证研究方法，完整的案例分析通过确立研究问题、选择研究案例和收集数据三步研究真实场景中出现的现象，适用于真实环境为对研究主体产生影响的因素之一、又或是实验数据时间跨度较大的场景。对于针对某些现象的初步调查，可使用探索性案例分析（Exploratory case studies）以提出新猜想和构建理论；而验证性案例分析（Confirmatory case studies）则用于验证现存理论。本文的案例分析有用于提出新猜想的探索性案例分析，也有为研究发现提供案例实证的验证性案例分析。

## 3) 调查研究 (Survey Research)

研究调查常用于需要从广泛个体中提取出普遍特征的场景，在数据收集上常以问卷调查为形式进行。然而，其所用的数据也可以通过数据日志技术或机构化访谈获取。对调查研究而言，最大的挑战是对采样偏差（Sampling bias）的控制。若采样出现偏差，所获数据无法代表目标群体，对应的结论也会失去代表性。根据以上定义，本研究可归入调查研究范畴之中，针对缓解采样偏差采取的具体措施，可见下一小节中外部有效性部分的讨论。

## 4) 社会学意义研究 (Ethnographies)

社会学意义研究通过实地考察的形式，对社群（Community）中的人们进行的社会活动进行理解。在软件工程领域中，这样的方法可帮助研究者了解应如何在社群中建立实践与交流文化，从而促进社群中的技术合作。社会学意义研究是把研究重点聚焦于人而非技术的一种研究方法，有助于揭示特定社群（在软件工程场景下为技术社群）的真实状况，从而为实践工作提供微妙而重要的见解。

## 5) 混合方法途径 (Mixed-method approaches)

混合方法途径指结合定量分析与定性分析，对研究对象进行系统数据解读的实证研究方法。按照实施的方式，混合方法途径可分为顺序解释策略（Sequential explanatory strategy）、顺序探索策略（Sequential exploratory strategy）与并发三角策略（Concurrent triangular strategy）三类。顺序解释策略先收集与分析定量数据，再收集和分析定性数据，以定性数据结果帮助解释定量结果；与之相反，顺序探索

策略先收集与分析定性数据，再收集和分析定量数据，以定量数据结果帮助解释定性结果；并发三角策略则会同时采用不同方法，以试图确认、交叉验证或证实已有发现。

### 2.2.3 有效性验证标准

除上述方法外，文献<sup>[70]</sup>还指出研究报告中应以一定篇幅分析实验设计或采取方法中可能使结论有效性受影响的部分（即有效性威胁，Threats to validity），以表示研究者在研究过程中已经考虑到了这些因素的影响，亦曾试图将此类因素对结论有效性产生的影响最小化。分析可从四个有效性验证标准出发进行，本研究在进行时对各标准进行了一定参考。

#### 1) 结构有效性（Construct validity）

结构有效性关注理论结构是否被正确解释以及相关指标是否被正确测量。为保证本研究的结构有效性，本文将在从各角度分析仿冒应用时，解释该维度用到的测量标准，以提供参考。

#### 2) 内部有效性（Internal validity）

内部有效性关注研究设计，要求研究者排除非自变量因素带来的影响。本研究的研究主体为Android中的仿冒应用，各实验中的自变量均为与仿冒应用相关的数据。从此角度分析，确保收集到的仿冒样本真实有效，即可在一定程度上确定本研究的内部有效性。使用应用证书中的信息鉴别仿冒应用是行之有效的，因此能保护本研究的内部有效性。

#### 3) 外部有效性（External validity）

外部有效性关注研究得出的结论是否具有普遍性，通常与研究中数据收集时的采样相关。考虑到结论的普遍性问题，本次研究中进行数据采集时，已尽可能广泛地进行采样。本研究的数据来自29个应用来源，50种目标应用涵盖11个类别，并一共采集到近14万个应用样本，由此可认为本研究的结论具有一定普遍性与代表性。

#### 4) 可靠性（Reliability）

可靠性关注其他研究者是否能通过重复报告中的实验复现报告中的结果与结

论。本研究的结果与结论均通过对数据进行分析获得，分析方法亦将于后文细述，其他研究者可利用文末提供的信息下载对应数据，再采用相同方法进行分析，可获得同样的结果。

### 2.3 本章小结

本章详细地阐述了 Android 的签名证书机制，简要介绍了重打包技术与 Android 的权限系统，为下文对仿冒应用的收集与筛选和后续的分析做铺垫。之后，本章介绍了实证研究的相关背景与前人提出的方法与建议。这些方法在本研究中起到了一定指导作用。

## 第三章 结合实例分析的仿冒应用特征解读

上一章节介绍了仿冒应用收集框架 FakeRevealer 的设计与实现。利用 FakeRevealer，本文顺利搜集到了近 14 万个应用样本。之后，本文利用这批数据对仿冒应用进行了全方位的挖掘分析。本章将结合实例详述本挖掘分析的详情，对仿冒应用进行特征解读。

### 3.1 研究概况

业界对恶意应用的理解使针对恶意应用的监控得以被实现，但与恶意应用同属移动黑灰产的仿冒应用却仍未被研究过。针对业界对仿冒应用了解匮乏的问题，本文利用了数据分析挖掘的方式，结合案例分析，完成了首次基于 Android 系统仿冒应用的特征解读。本研究定义的三个不同视角分别为：仿冒应用的基本特征、影响仿冒应用数量的因素和仿冒应用的发展轨迹。

### 3.2 仿冒应用的基本特征

为了了解仿冒应用开发者使用的仿冒策略，或者说，他们是如何绕过应用市场的监管机制的，研究者有必要对仿冒应用的基本特征有所了解。为此，本研究分别针对应用的安全证书和应用名称、包名和应用大小等基本信息进行了观测。

应用的安全证书就是对开发者的识别码，而仿冒应用在安全证书上的性质（也就是说，是否每个仿冒应用都有一个独一无二的安全证书），十分可能是仿冒应用规避监管技术的关键。另一方面，本文还猜想重打包应用在本研究的数据集中普遍存在，而对仿冒应用基本信息的测量（比如说对应用包名和大小的测量）有助于了解重打包应用在数据集中的分布如何—因为普通的重打包技术并不会对 APK 的基本信息（比如应用名、应用版本号等）作出修改，除非仿冒应用开发者有意为之。

### 3.2.1 安全证书与仿冒应用的对应关系

那么，安全证书和仿冒应用的对应关系如何？对此，本文提出了假设：

**假设 1.1：**绝大部分的仿冒样本有其对应的、独一无二的安全证书。即绝大部分仿冒应用和他们的安全证书呈一对一的映射关系。

与之对应，本文提出了以下的研究问题：

**RQ 1.1：**仿冒样本数量和他们的安全证书数量存在着什么样的关系？也就是说，一个安全证书通常会跟多少个仿冒应用样本相关联？

为了解答这个问题，本文从所有搜集到的样本中提取出安全证书，然后对这些证书和样本做了配对。根据配对得出的数据，本文得出了以下结果：

**RQ 1.1. 结果** 在仿冒应用持有的所有安全证书中，76% 都仅仅关联到了一到两个仿冒样本，与单个安全证书相关联的仿冒应用数分布在从 1 到 1,374 的区间内。表 3.1 统计了安全证书和他们对应的仿冒样本数量。其中第一栏为仿冒样本的数量区间，第二栏为关联的仿冒样本数量该落于区间的安全证书数。大部分安全证书都只关联了 1 到 5 个应用样本，但也有少量安全证书与大量仿冒样本有关联关系。

表 3.1：安全证书/仿冒应用数量对应表

仿冒样本数量	1-5	6-10	11-50	51-100	More than 100
安全证书数量	8252	525	531	71	80

这个发现与本文的猜想（即大多数仿冒样本都有他们对应的独一无二的安全证书）部分相符。虽然并不是大多数仿冒样本都有其单独对应的安全证书，但多数安全证书的确只对应少量样本。结合第 2.1.1 节 Android App 签名机制最后的说明，本文认为这是规避应用市场监管机制的一个策略。如果以同一开发者的身份，用一个安全证书上传多个仿冒 App，万一其中 App 被投诉下架，其他的 App 很可能会受到牵连。然而，一个仿冒开发者其实也可以持有多个安全证书。如果使用多个安全证书分别上传仿冒 App，应用市场就不容易找到这些 App 的关联，即使其中一部分被举报下架，余下的也得以被保全。另外，在整理对应多个仿冒样本的安全证书时，本文得到了一些意料之外的发现。相关内容会在后续案例分析中加以拓展。

### 3.2.2 仿冒样本与原版应用的相似度

除了安全证书以外，应用的外观也是十分重要的基本特征。鉴于运算量等原因，本研究暂时无法将应用图标和应用内布局等因素用于仿冒样本和原版应用的对比，但本工作依然提取出了样本的包名/应用名/APK 包大小等最基本的项目数据以比较仿冒样本和原版应用的相似度。在这个方面，本节的假设如下：

**假设 1.2：**一大部分的仿冒样本和他们的仿冒对象（也就是原版的官方 App）有同样的应用名/包名/APK 包大小。

与上一假设类似，本假设也对应一个研究问题：

**RQ 1.2：**在本研究的数据集中，仿冒应用是怎么“山寨”正版 App 的？也就是说，仿冒应用的应用名/包名/APK 大小和他们对应的正版 App 有多相似？

以编辑距离作为度量，本研究对目标应用和仿冒样本的应用名/包名相似度作了比对，同时对比了两种样本的大小区别，得出了以下结果：

**RQ 1.2. 结果** 以包名为例，根据本研究的统计结果，在所有的 52,638 个仿冒样本中，只有 243 个（少于 0.5%）使用了正版应用的包名，余下大于 99.5% 占比的所有仿冒样本都使用了他们自定义的包名。在余下的这 52,395 个样本中，本研究找到了 14,089 个不同的包名。但这其实是意料之中的结果。因为每个应用在 Android 系统中都需要有独一无二的包名，如果系统在安装 App 时发现系统中已经有具有相同包名的 App，就会检查两个不同版本应用的安全证书，证书不一致会导致安装失败。因此大部分仿冒应用不会直接使用正版 App 的包名。但这是否意味着仿冒应用就会使用与正版 App 完全不同的包名呢？它们会不会使用和正版 App 相似的包名？

同理，本节对应用名和 APK 大小两个方面也有类似的疑问。下面就本节获得的数据，探索上述问题。

为了解决相似度问题，本文采用了编辑距离<sup>[71]</sup>这一在自然语言处理（Natural Language Processing，简称 NLP）领域被广泛应用的距离定义作为衡量标准。

#### 定义 1 编辑距离

给定两个字符串  $a$  与  $b$ ，其间的编辑距离  $d(a, b)$  为将  $a$  和  $b$  相互转换的最小编

辑操作数。其中，每次添加、删除或将一个字符转换成另一个字符都算作一次编辑操作。

举个例子，“jingdong”和“jindeng”之间的编辑距离是 2，由前者转换为后者的其中一种编辑次数最小方法是将第一个“g”删除，然后再将“e”转成“o”。同理，字符串“fake”和“official”之间的距离是 7，其中一种方案是在“f”前添加“o”，在“f”和“a”之间添加“fici”（这里包含了 4 步操作），将“k”替换成“l”，最后删去“e”。对于从仿冒样本中获取到的每个包名，本文都计算了与其对应的官方发布 App 的正版包名的编辑距离。

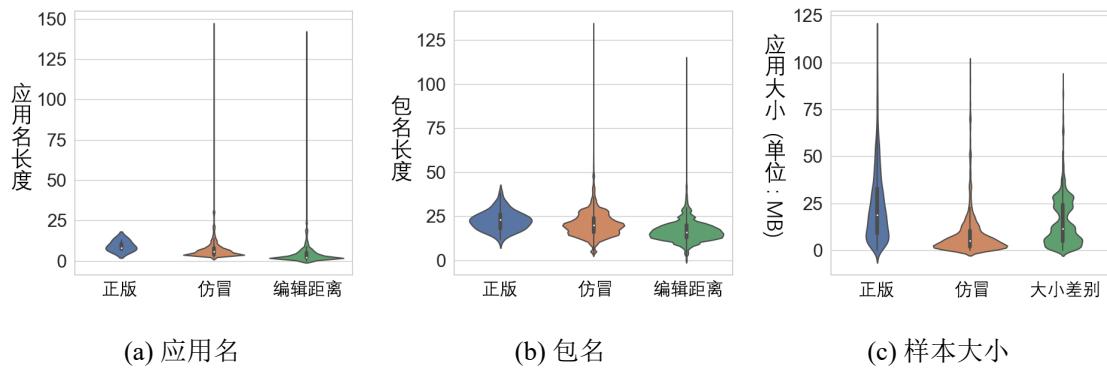


图 3.1: 对 App 各项属性的统计结果

图 3.1 由三个小提琴图<sup>[72]</sup>组成，分别表示了本文在应用名、包名和 APK 包大小上的统计信息。在每个“小提琴”中，中间的黑色粗条表示四分位数范围，粗条中间的小白点表示数据的中位数，而黑色细条表示 95% 置信空间。

3.1a展示了分别在官方样本、仿冒样本的应用名和两者间编辑距离的统计数据。其中“官方”图例和“仿冒”图例中的小白点都在接近数值“6”的位置，说明官方样本和仿冒样本的应用名的平均长度十分相近。这两个数据组之间的数据分布的主体略微相近，也表面了他们的相似程度。更重要的是，图中“编辑距离”图例的中位数值十分低（在  $y$  轴上为“2”的位置）。这意味着过半数仿冒应用通过从官方 App 的应用名中修改少于 3 个字符来获得其应用名。这表示大部分仿冒应用正在使用与官方 App 非常相似的应用名。与此同时，本研究也留意到了一些仿冒应用有着异乎寻常的长名称（其中最长的仿冒样本的应用名中甚至有 146 个字符）。

本研究认为这些异常样本有可能是出于测试市场审查机制的目的而被上传到市场上的。另一种可能是，一些长应用名拼合了多个热门 App 的名称（比如“潮流女装-美丽说蘑菇街淘宝天猫京东美团精选”），这可能是为了应用更容易地被用户搜索到而采取的策略。

3.1b 显示了针对包名的结果。和 3.1a 类似，官方 App 的平均包名长度和仿冒样本的平均包名长度依然很类似（双方的值分别为“23”和“20”）。然而，他们之间编辑距离的中位数则比应用名编辑距离的中位数明显更高（在  $y$  轴上“16”的位置），这意味着将一个仿冒应用的包名转换为一个官方 App 的包名平均需要 16 次修改，反之亦然。也就是说，官方 App 的包名和仿冒应用的包名会相当不一样。可以据此推出仿冒应用更倾向于使用自定义的包名。

3.1c 显示了 APK 包大小的信息。为了能更好地显示结果，一些极端样本在作图前被剔除出数据集：他们是大于 150MB 的 APK 包，在所有 69,614 个官方 App 的样本中占 851 个（约为 1%），在 52,638 个仿冒样本中占 447 个（少于 1%）。这些极端样本大部分来源于“游戏”类别下。图表显示，仿冒样本大小的中位数约为 5MB，而约半数的正版 App 有着大于 18MB 的大小。因此，仿冒应用更有可能：

1) 由仿冒开发者自行开发，而不是使用重打包技术制作，因为重打包之后的应用通常不会在大小上与原版有太大差距；

2) 是恶意应用，因为恶意应用除了恶意代码之外，通常没有太多其他内容。

简而言之，图 3.1 说明了仿冒应用：

1) 更倾向于用一个与正版 App 相似（甚至是相同）的应用名，但会使用自定义的包名；

2) 通常大小更小。

在很大程度上，本研究认为产生第一点结论的原因是应用市场上提供的 App 信息的缺失和用户对 Android App 了解的缺乏。如图 3.2 所示，在应用市场上，当用户浏览 App 的详情页时，能看到应用名、下载量、应用描述、其他用户对应用的评论和评分等信息。但是，大部分应用详情页上并不会出现技术详情，比如应用包名。而普通用户也不会了解 Android App 中包名和应用名的区别和关联。个别应用市场（如 3.2d 所示的小米应用市场）上，应用的某些信息甚至被折叠了起来，用户

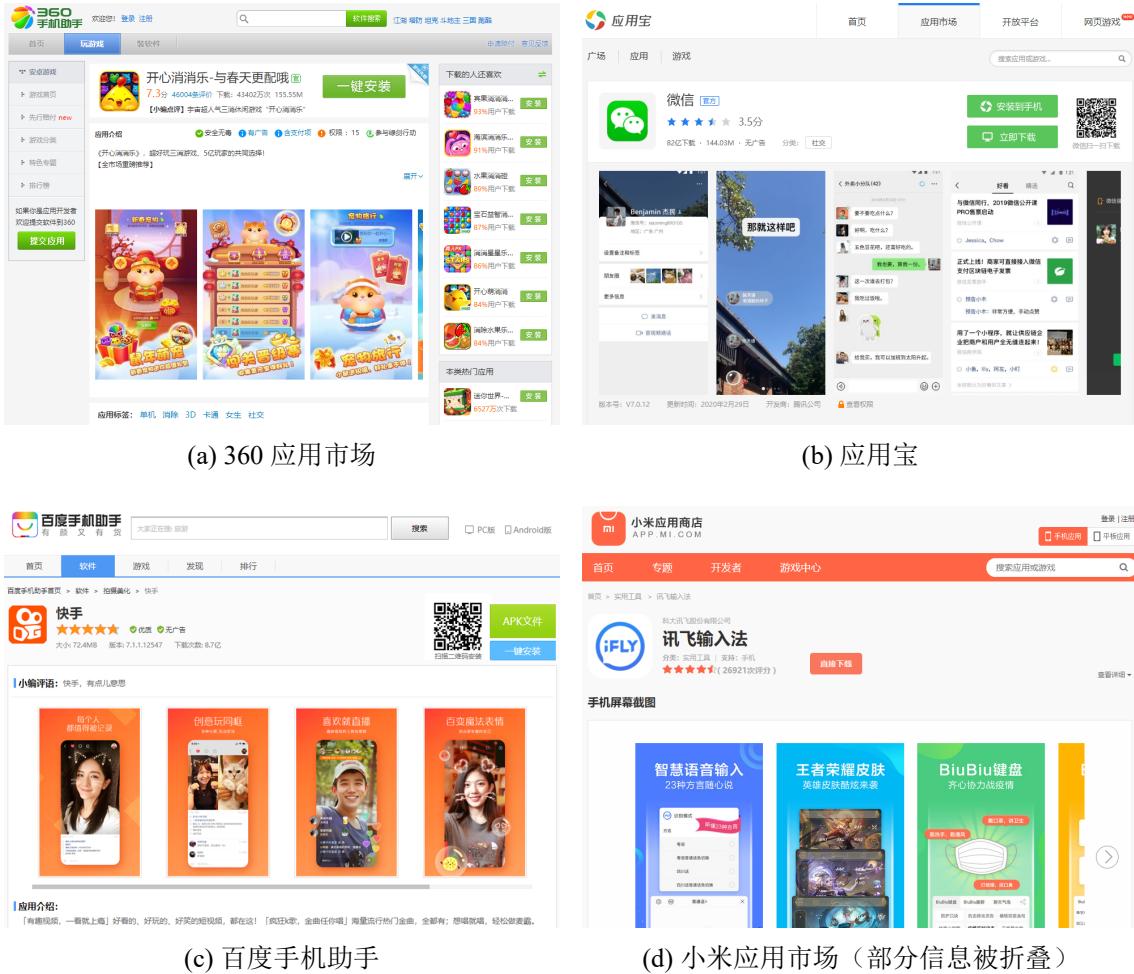


图 3.2: 各大应用市场应用详情页

要点开折叠页才能发现一个 App 会有多大。因此，不会被普通用户关注的包名自然可能会出现各种五花八门的情况，而一定会显示的应用名则是越接近正版 App 越容易诱导用户下载。

### 3.2.3 案例 1. 持有官方安全证书的可疑样本

在人工浏览数据时，本研究发现了一个具有可疑应用名的样本—该样本声称自己是一个“破解版”的应用。因此，本研究针对所有 69,614 个持有官方证书的样本进行了以下筛选：

1. 使用“破解”、“免费”等关键字搜索所有带正版证书的样本，筛选出带有可疑应用名的样本；

2. 筛选出所持安全证书与原开发者不一致的样本；
3. 筛选出包名和同款 App 的多数样本不一致的样本。

最终，本文获得了 17 个由正版开发者安全证书签署的可疑样本，其中三个样本的信息如表 3.2 中所示，分别代表上述三项筛选得到的结果。第一个名为爱奇艺的样本虽然由一个官方安全证书签名，但该安全证书和其他爱奇艺样本的却不一样。对比之后，本研究发现该证书来自 360 手机助手，但 360 和爱奇艺并没有合作关系，因此这是个可疑的样本；而第二个样本（360 手机助手）的可疑之处在于样本包名。多数 360 手机助手的包名为 *com.qihoo.appstore*，也有少部分官方包名为 *com.qihoo.secstore*，前者为 360 手机助手在国内第三方应用市场发行的应用包名，后者为 Google Play 官方应用市场上上架的包名。然而，其中一个使用了其官方安全证书签署的样本的包名却是 *com.kuyou.sdbgj.baidu*，十分奇怪；第三个样本则是在应用名中包含了“破解”字样。然而，正常的正版应用根本不会有这样的命名方式，所以本研究也认为这是一个可疑样本。

VIRUSTOTAL 的检查结果显示，17 个可疑样本中，只有 2 个是良性应用，2 个是 PUP，余下 13 个样本都被判定具有恶意行为。进行后续分析时，相关样本已被剔除出正版样本集合。

表 3.2: 持有官方安全证书的可疑样本

样本应用名	样本 SHA1 码	可疑之处
爱奇艺	b86c55a509e8293b24138b166e9ff410f39e84b5	可疑证书
360 手机助手	2bb43c53b86d204d0040a8af6cb2a09cf9e93bb7	可疑包名
Youku XL 破解版	b55b7ef189d649aeb03443c5d1ab57c9031d624e	可疑应用名

鉴于这 17 个样本都是持有官方安全证书签名的，有理由怀疑有应用厂家不慎泄露了自己的安全密钥库，从而导致了这些样本的出现。然而，如果真的是因为厂家泄露密钥库，一来很有可能会导致恶意开发者使用官方安全证书大量生产恶意应用，二来对应厂家也会出于安全考虑马上更换新的包名和安全证书。本研究的数据并不支持以上猜想带来的两点结果，所以本文不认为这是由于安全证书泄露导致了这些可疑样本的产生。

除去这个可能性，本文认为更有可能的原因是某些仿冒应用开发者掌握了穿透/绕过 Android 系统签名机制的技术，从而产生了这些样本。

时间回溯到 2017 年 12 月，Google 确认并公布了 V1 版本应用签名机制的一个后门（CVE-2017-13156）<sup>[73]</sup>。通过这个后门，黑客可以在不修改 APK 包安全证书信息的情况下，向 APK 包里注入任意内容。而早在这个漏洞被公布的至少一年之前，Google 就已经发布了作为 V1 版签名机制的替代解决方案，也就是 V2 版应用签名机制。这看起来十分有可能是导致这些样本产生的原因，某些恶意开发者利用了 V1 版签名机制的漏洞，修改了 APK 包的基本信息。为了确认这些样本是否采用了具有风险的 V1 版应用签名机制，本文使用了 apksigner 来检测这些样本使用的签名机制版本。apksigner 是 Google 官方提供的一个命令行工具，它被集成在 Android SDK 中，既是 APK 包编译打包过程中为 APK 包进行数字签名的工具，也可以用来验证 APK 包使用的签名机制版本，又或者是验证 APK 的签名是否有效。

apksigner 的结果显示，所有 17 个样本都只使用了 V1 版本的应用签名机制。在了解到 V1 版本签名机制已经不再安全的情况下，仍有部分开发者由于各种原因没有接受更新也更安全的签名方案，这个结果令人失望。

**本节小结：**绝大部分安全证书只与少数仿冒样本有所关联。这很可能是仿冒开发者规避市场监管机制采用的策略。同时，本文也观测到仿冒应用倾向于于官方 App 相同或者是十分相似的应用名。但是，仿冒样本和官方 App 在包名和 APK 包大小方面都不相似，这表明重打包应用在仿冒应用中并不普遍存在。最后，如果良性应用的开发者不遵从最新的安全标准发布应用，可能会导致十分严重的安全问题。

### 3.3 影响仿冒应用数量的因素

天下熙熙，皆为利来；天下攘攘，皆为利往。不妨假设获利是驱动仿冒应用开发者的最终目标，进而假设仿冒应用的数目与其来源市场、受欢迎程度及其应用分类密切相关。因为一个应用越受欢迎，就越可能获利，对仿冒应用也是如此。此外，App 的更新频率也可以被视作一个潜在因素，频繁更新的 App 或许会阻碍仿

冒应用开发者对其进行仿冒。

### 3.3.1 仿冒应用的来源

本研究中搜集的应用样本来源于多个不同的应用市场，每个市场架上的应用数量不一，其审核、监管力度也并不一致。这引出了一个问题：应用商店架上的应用样本数量是否会跟仿冒应用的数量有关系？为此，本节有了以下假设：

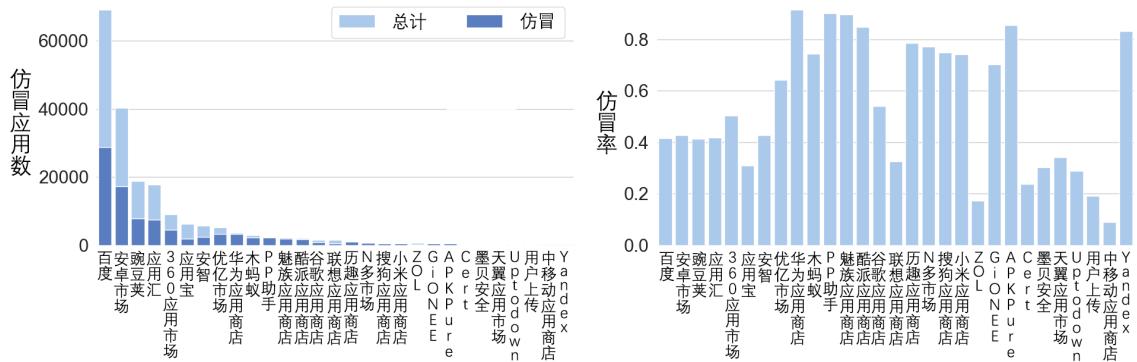
**假设 2.1：**仿冒样本的比率与应用市场架上的 App 数量有关联。

与之相对，本节有研究问题如下：

**RQ 2.1：**这些仿冒应用市场都集中来源于哪里？哪个应用市场有最多的仿冒应用？

根据各个样本的来源，本节从数据统计角度分析了结果。

**RQ 2.1. 结果** 图 3.3 展示了本研究收集到的所有样本的来源。左图显示，在本研究收集数据的所有 29 个应用来源中，源于百度手机助手的样本量是最大的。同时，从百度手机助手中搜集到的仿冒样本也是最多的。各个应用市场的仿冒率在右图呈现。



$$fake\ sample\ rate_a = \frac{fake_a}{total_a} \quad (3.1)$$

$$fake\ sample\ rate_{AS} = Avg(fake\ sample\ rate_a, \forall a \in \{\text{目标 App}\}) \quad (3.2)$$

尽管百度手机助手<sup>[74]</sup>和安卓市场<sup>[75]</sup>的仿冒率均为40%左右，在所有的31个渠道中处于中等水平，但由于源于这两个渠道的样本基数最大，所以从这两个应用市场搜集到的仿冒样本数也是最多的。

图中数据显示，应用市场的样本数量和仿冒率并没有直接联系，但是本节仍然有一个有趣的发现，那就是App本身和市场的关系有可能是影响App仿冒率的一个因素。腾讯旗下的应用市场应用宝<sup>[76]</sup>中较低的仿冒率可以很好地为这个发现提供数据支持，因为本研究的50款目标App中，有12款都是腾讯公司开发的应用。

### 3.3.2 其他因素对仿冒样本数量的影响

除了市场本身之外，本节还假设以下这些因素可能会影响某款App对应的仿冒样本数量：

**假设 2.2：**仿冒应用的数量与其对应的正版App的受欢迎程度有密切联系。

**假设 2.3：**应用的更新频率影响着其对应仿冒应用的数量。

**假设 2.4：**App类别是影响仿冒应用数量的因素之一。

与三个假设对应的是三个研究问题：

**RQ 2.2：**一个App受欢迎的程度会影响对其仿冒的应用数量吗？

**RQ 2.3：**一个App的更新频率会影响对其仿冒的应用数量吗？

**RQ 2.4：**一个App所在类别会影响对其仿冒的应用数量吗？

对于这三个研究问题，本节使用了皮尔逊积矩相关系数（Pearson product-moment correlation coefficient，简称PPMCC）来衡量应用数量和问题对应的几个维度的关联性。

**RQ 2.2. 结果** 从直觉上看，某款App越受欢迎，仿冒应用开发者就越有动机对其仿冒，然后诱导用户下载仿冒版本以获取利润。

要注意的是，每款目标 App 都有不同的样本数（无论是官方的或是仿冒的），所以在这里不能拿仿冒样本的数量直接作比较。为了消除偏差，本文将样本数量归一化，使用式 3.1 定义的仿冒率对每个目标 App 进行比较。接下来，本文使用了皮尔逊积矩相关系数来计算一款 App 被仿冒的严重程度与其热度是否具有相关性。相关计算会使用上述的仿冒率和从易观千帆<sup>[77]</sup> 获取的 App 月度热度指数计算。

### 定义 3 皮尔逊积矩相关系数

两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商（式 3.3）。

$$p_{x,y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} = \frac{E[(X - u_x)(Y - u_y)]}{\sigma_x \sigma_y} \quad (3.3)$$

式 3.3 的值域为  $[-1, 1]$ ，该值越接近 0，表示两个变量之间的相关关系越弱。出人意料的是，数据显示，两个因子之间的相关系数只有 0.246。这表明仿冒应用的数量和 App 的热度在相关性上只处于较弱水平，和本研究的预期并不符合。

**RQ 2.3. 结果** 本研究猜想更新频率有可能会与 App 被仿冒的次数相关联，因为升级通常会有漏洞修复等举措，可以帮助 App 免受攻击。所以一款 App 的更新频率越高，其安全性能应该就会越好。

为了评估一款目标 App 的平均更新频率，本研究标记了每个官方应用样本被发行时的时间，精确到日。然后，再找到最新发布的那个样本和最早发布的那个样本，求出他们发行时间的差值的绝对值与版本数的商，即为平均更新频率（单位：天/版本）

相关系数的计算结果表示，更新频率和仿冒数量之间的关联度只有 0.084，意味着两者之间几乎没有关联。本研究认为这个结果由两个原因导致：

- 1) 在本研究的数据集中，每款 App 的平均更新间隔为 10 天/版本。这个较高的更新频率表面开发者可能不会每次都在更新中修正安全性问题，从而削弱了更新频率作为安全性指标的功能。
- 2) 结合前文的结果，数据集中的大部分仿冒样本都不是重打包应用，而是仿冒应用开发者自行开发的。因此，无论官方版本受到的保护程度如何，仿冒应用开

发者都可以制造出对应的仿冒应用。

**RQ 2.4. 结果** 某些 App 类别比其他类别更有可能带来收益。根据一份来自 App 营销机构 LIFTOFF<sup>[78]</sup> 的报告预测，在未来，游戏类有望成为带来最高收入的应用分类。

表 3.3: 目标 App 与其相关统计

应用名	类别	月度热度指数	更新频率 (天/版本)	样本总数	仿冒 样本数	仿冒率	平均仿 冒延迟
微信*	社交网络	91.2K	6.4	9248	6447	69.7%	12.1
QQ*	社交网络	54.6K	10.7	11167	3780	33.8%	9.2
爱奇艺	视频	53.5K	6.4	7586	3481	45.9%	9.3
支付宝	生活	48.1K	10.2	983	231	23.5%	10.1
淘宝*	移动购物	47.5K	7.0	6003	3010	50.1%	8.1
腾讯视频	视频	47.3K	6.3	1429	68	4.8%	10.7
优酷	视频	40.9K	7.3	2058	262	12.7%	6.7
新浪微博*	社交网络	39.2K	5.3	5947	2715	45.7%	5.7
WiFi 万能钥匙	系统工具	36.4K	3.1	4808	2999	62.4%	3.0
搜狗输入法	系统工具	33.3K	11.0	898	40	4.5%	21.8
百度	资讯	32.4K	11.1	15651	3514	22.5%	12.8
腾讯新闻	资讯	28.7K	8.5	1051	11	1.0%	8.9
QQ 浏览器	资讯	27.8K	5.6	1369	43	3.1%	11.6
今日头条	资讯	27.4K	4.4	3538	179	5.1%	5.6
应用宝	应用市场	27K	11.4	2419	266	11.0%	11.6
快手	视频	24.4K	3.2	8273	4270	51.6%	3.5
腾讯手机管家	系统工具	24.2K	8.7	2463	1340	54.4%	8.7
高德地图	生活	24K	6.5	1225	51	4.2%	13.1
酷狗音乐	音乐	23K	8.6	1313	122	9.3%	12.2
QQ 音乐	音乐	21.7K	9.4	1132	65	5.7%	14.6
百度地图	生活	21.3K	8.8	2609	1438	55.1%	15.3
抖音	视频	19.4K	11.1	317	12	3.8%	8.3
京东*	移动购物	18.5K	10.9	5000	2377	47.5%	12.3
UC 浏览器	资讯	16.7K	7.4	4232	1624	38.4%	7.0
360 手机卫士	系统工具	15.4K	12.4	3670	1423	38.8%	19.1
全民 K 歌	音乐	14.7K	21.1	618	215	34.8%	17.3
美团	生活	13K	8.0	4752	1415	29.8%	6.9
拼多多*	移动购物	12.9K	6.6	2327	551	23.7%	7.8
王者荣耀*	游戏	12.5K	15.5	2350	1319	56.1%	12.3
美图秀秀	摄影录像	12.4K	5.4	1705	784	46.0%	5.8

火山小视频	视频	12.2K	11.9	410	16	3.9%	9.6
墨迹天气	生活	12K	4.2	10081	7093	70.4%	4.7
滴滴出行	生活	11.8K	8.6	943	117	12.4%	7.0
华为应用市场	应用市场	11.8K	N/A	0	0	0.0%	N/A
<b>开心消消乐*</b>	游戏	<b>11.2K</b>	<b>19.7</b>	<b>2406</b>	<b>1738</b>	<b>72.2%</b>	<b>20.6</b>
酷我音乐盒	音乐	11K	2.9	3778	69	1.8%	4.2
西瓜视频	视频	11K	11.5	866	100	11.5%	8.8
OPPO 应用商店	应用市场	10.8K	N/A	0	0	0.0%	N/A
猎豹清理大师	系统工具	9.9K	10.3	1803	388	21.5%	13.5
360 清理大师	系统工具	9.6K	17.3	327	8	2.4%	8.5
360 手机助手	应用市场	9.2K	7.6	1616	137	8.5%	8.4
WiFi 管家	系统工具	8.8K	19.5	1636	658	40.2%	15.7
讯飞输入法	系统工具	8.6K	6.0	1451	8	0.6%	10.1
百度手机助手	应用市场	8.2K	11.4	3849	437	11.4%	14.5
小米应用市场	应用市场	7.8K	N/A	0	0	0.0%	N/A
<b>WPS Office*</b>	商务办公	<b>7.4K</b>	<b>6.0</b>	<b>1152</b>	<b>69</b>	<b>6.0%</b>	<b>7.8</b>
美颜相机	摄影录像	7.1K	5.3	1600	691	43.2%	6.3
网易云音乐	音乐	7K	10.5	616	6	1.0%	12.2
网易新闻	资讯	6.7K	7.0	1441	93	6.5%	5.0
<b>QQ 邮箱*</b>	商务办公	<b>6.6K</b>	<b>16.4</b>	<b>520</b>	<b>11</b>	<b>2.1%</b>	<b>10.4</b>

\* 详情会在 **RQ 2.4.** 结果中给出

根据应用功能划分，本研究的 50 款目标 App 可以被分为 11 个类别。表 3.3 按每款 App 的热度排序，展示了每款 App 的类别和他们的仿冒率，同时还有他们的更新频率、关联样本总数等数据。在相同的类别下，多数应用之间的仿冒率差值都位置在一个合适的水平。毫无疑问地，娱乐方向的类别（比如游戏和社交网络）吸引了更多仿冒应用开发者对其仿冒。而另一个类别移动购物也特别受到了仿冒应用开发者的“关照”，因为移动购物在中国也正在快速地发展。相对来说，商务方向的商务办公类的应用就不是那么地吸引仿冒应用开发者了，这个领域下的 App 平均仿冒率只有 4.05%。上述四个类别的应用都在表中被加粗标识，读者可以自行查阅。

这个结果与日常生活中可观察到的结果相符，比起商务类用途，普罗大众更倾向于使用移动设备用作娱乐用途，消遣时间。仿冒应用的数量从某种角度上反映了人们在日常生活中如何使用移动设备，这是个十分有趣的发现。

### 3.3.3 案例 2. 游戏类别下的仿冒应用

正如表 3.3 中数据所示，游戏类应用（王者荣耀和开心消消乐）吸引了大批的仿冒应用样本。出于性能考虑，此处的数据中只提取了 APK 包的基本信息，并没有对收集到的每个 APK 文件进行详细的分析。因此，为了弄清楚这些仿冒应用究竟是怎么样的，本研究随机从这两款游戏 App 的仿冒样本中选择了一些样本（每款应用选择 7 个仿冒样本），然后将这些样本安装到了实验设备上。

3.4a 展示了这些样本在真实的 Android 设备上安装之后的实际外观。官方渠道下载的正版 App 在图片中由绿色边框标记出。明显地，与官方应用相比，仿冒应用或者有一个和官方应用名十分相似的应用名，或者在图标上和官方相近甚至相同。



图 3.4: 游戏类 App 及其仿冒样本

本研究在测试设备上实际运行了上述安装的 14 个仿冒样本，然后拿它们和原版的应用对比。测试使用的设备为高配版小米 5 手机，搭载的 CPU 为最高主频 2.15GHz 的骁龙 820 处理器，3GB 内存，64GB 机身存储，安装的 Android 系统版本为 Android 6.0 (Android Marshmallow, API 23)。

3.4b 和 3.4c 分别是在测试设备上运行官方版本的开心消消乐和其中一个仿冒版的开心消消乐时的系统截屏。不难看出，两款应用的外观是十分相像的。本研究

在测试时发现，两款游戏内部的玩法、实际操作逻辑也一模一样。如果不是事先知道了哪一款应用是来自官方渠道下载的正版，一般的应用开发者也未必可判别两个应用的真伪，更不必说是从应用市场搜索结果中找到这些结果的普通用户了。

而这并不是唯一的案例。作为结果，本研究发现 7 款开心消消乐的仿冒样本中，有 4 款是与官方样本十分相似的游戏（其中一个十分可能是经过重打包技术处理的应用），2 款声称自己是“系统攻略”，还有 1 款在运行时闪退，无法在测试设备上实际运行。在 4 款仿冒游戏中，3 款都在游戏中不时自动弹出游戏内购窗口，要求玩家购买道具，十分可能导致玩家不想要的花费。而所有 7 个仿冒样本都在 VIRUSTOTAL<sup>[79]</sup> 中被报告为恶意应用。

相比之下，王者荣耀的仿冒样本内容就与官方应用大相径庭了。在 7 款被安装到测试设备的仿冒样本中，有 3 款是壁纸浏览器，里面包含了几张游戏内人物的插画，可以在应用内将这些插画设置成系统的桌面壁纸；而余下四个是简单的拼图游戏，里面同样包含了王者荣耀游戏人物的插画，应用内容就是简单地把被打乱的插画拼图恢复原状。Virustotal 的结果显示，7 款仿冒样本中，有 6 款是恶意软件，涵盖了木马病毒、广告软件等类型，而余下的一款则被报告为潜在有害程序（Potentially Unwanted Program，简称 PUP）。PUP 通常在用户不知情或者不愿意的情况下，通过静默安装或者捆绑安装的形式被安装在系统中。尽管这种软件不一定包含恶意代码，但其动机十分可疑。

本研究认为，这种现象是由模仿正版应用功能的难易程度带来的。只从技术角度看，像王者荣耀这样的多人在线战斗竞技场（Multiplayer Online Battle Arena，简称 MOBA）游戏核心难度明显要比开心消消乐这样的益智类游戏要高得多。一款 MOBA 游戏除了要解决支持运行运行的物理引擎之外，还要实现聊天系统、在线匹配、负载均衡等业务，更加不必说背后的人物设计技能平衡等更深入的话题了；而一款益智类三三消游戏的核心逻辑就只在于元素三连的判定和随机新出现的元素，再加上道具系统就差不多可以包装成一个完整的游戏推出。

因此，就算不考虑后续的维护问题，要开发一款像王者荣耀这样的复杂游戏，对仿冒应用开发者来说明显是成本过高的。但由于这款游戏本身具有超高的热度，可能带来巨大的收益，所以仿冒应用开发者会为了蹭上热度而开发外观相似、内

容完全不符的仿冒样本。相比之下，开心消消乐由于开发难度相对较小，所以仿冒应用开发者会愿意开发一个内容相似的应用，再通过内购陷阱等手段收取效益。这两款 App 透露出了仿冒应用开发者在仿冒方面两个截然不同的思路。

本文在第 3.3 节中观察到的商务办公类别有较低仿冒率也可能是类似原因导致的结果。一方面，商务办公类的工具核心逻辑比较复杂，对仿冒开发者来说并不是一个有利可图的最佳选项；另外，这类应用也不像游戏一样会衍生出周边产品（比如王者荣耀的游戏人物就会有不少插画），仿冒应用开发者也没办法从这方面入手蹭热度。结合两个原因，商务办公类的应用自然不会引起仿冒应用开发者的太多兴趣。

**本节小结：**正如由本文的统计和计算揭示的那样，从一个应用市场中能找到的应用样本数量并不能与应用商店的仿冒率相对应。此外，App 本身与应用市场的关系也会影响到市场中对应 App 仿冒样本的数量。出人意料的是，App 的更新频率并没有与仿冒率相关联。本研究认为这是由于应用更新频率太高、而且重打包应用在本研究的数据集中占少数而导致的结果。本文进一步观测到，与更新频率和应用热度相比，应用分类这一因素对 App 的仿冒率有更大的影响。案例分析从游戏类别的仿冒应用入手，说明了仿冒应用开发者对不同应用会采用不同仿冒策略。

## 3.4 仿冒应用的发展轨迹

在这个视角下，本文希望结合时间维度，从本研究的数据中挖掘信息。随着时间推移，仿冒应用的特征和行为模式是否有发生改变？这些年来，仿冒应用这个灰色产业是否有过变迁？利用本研究提取的数据中的搜集时间数据项，本文复原了各种不同的时间线以解答上述问题。

### 3.4.1 仿冒应用的研发延迟

在正版应用推出新版后，如果一个仿冒应用能在越短时间内推出对应的新版本，仿冒应用的开发者就越有可能蹭上软件更新的热度，从而获利。对此，本节提出了以下研究问题：

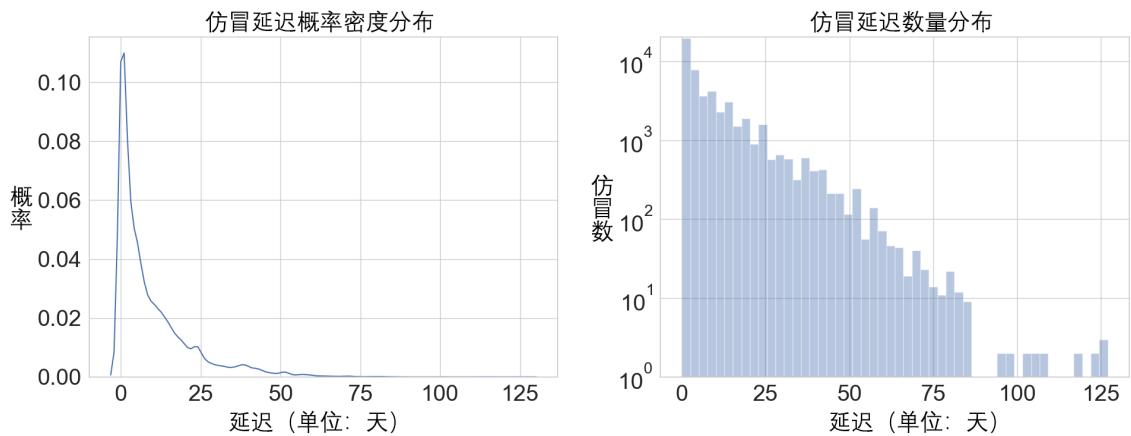


图 3.5: 仿冒延迟总体分布

**RQ 3.1:** 在一个官方应用的新版本推出之后，仿冒应用开发者需要花多少时间去推出对应版本的仿冒版？换句话说，这些“山寨”版本会过多久出现？

在分别复原官方应用的更新时间线和对应仿冒版本的发布时间之后，本文得到了以下结果：

**RQ 3.1. 结果** 本文计算出了每版 App 被仿冒的延迟时间和延迟的分布情况，结果显示在图 3.5 中。

出于多种原因，本文很难为研究中的所有 50 款目标 App 回溯出它们每个版本更新的时间点，然而，凭借数据库中的数据，本文大致地重现了他们的更新时间线。首先，本文对所有收集到的官方样本按照他们的 App 源进行分类，然后再按照版本号对 App 源类下的所有样本分类。举个例子，一开始是使用“爱奇艺”作为搜索源的所有官方样本都会被归回“爱奇艺”类下，然后再按版本号分类。这么做的原因是，即使是官方渠道发布的版本，有些 App 在不同应用市场上架的时间和内容也会略微有差别，可能会导致一个版本的正版 App 有多个样本的结果。这个做法就是为了消除上述情况带来的影响。之后，对于每款 App 的每个版本，本文都按照对应样本被爬取到的日期时间戳对其进行排序。这样的话，通过提取每个版本中第一个版本被爬取的时间，就可以知道这个版本的官方 App 最早的发布日期了。最后，将每款 App 中每个版本最早发布的日期串联起来，就可以大致重现每款 App 的更新时间线。

由于仿冒样本大多不是重打包应用，本研究没办法为每个仿冒样本精确匹配

一个对应的正版版本，为了找到某个仿冒样本的仿冒延迟，本文提取出它被爬取的日期时间戳，然后将这个时间戳与正版更新时间线上的所有版本进行对比，找到不晚于这个仿冒样本发布的最晚发布官方版本的发行时间，然后取他们的时间差作为仿冒延迟。按照图 3.5 的结果，绝大多数仿冒样本在官方应用推出后的 20 天内就被发布了。根据本文的统计，有 60% 仿冒应用在正版 App 被推出的 6 天内就被发布了。这表明仿冒开发者的行动十分迅速。

### 3.4.2 仿冒应用安全证书的存活时长

从某个角度看，仿冒应用安全证书的存活时长反映了应用市场的监管力度大小，也能反映市场之间在安全方面是否具有良好的合作机制。所以本节有研究问题如下：

**RQ 3.2:** 一个仿冒应用开发者的安全证书可以存活多久？

本文整理了不同仿冒应用安全证书的出现时间，得出了下面的结果：

**RQ 3.2.** 结果图 3.6 展示了不同仿冒应用安全证书在应用市场里存活时间的总体分布。在左边的密度分布函数图中， $x$  轴表示其存活的时长， $y$  轴则表示了与  $x$  轴上的值对应的概率密度。曲线下的总面积为 1，任意两个  $y$  值  $y_1$ 、 $y_2$  之间的曲线下面积是其对应的  $x$  轴上的值  $x_1$ 、 $x_2$  在数据中占有的概率。比如说，图 3.6 中  $x$  轴从 0 到 200 之间的值对应的  $y$  轴曲线下方的区域面积接近 0.8，意味着约 80% 的仿冒应用安全证书不会存活多于 200 天。

断定一个仿冒应用安全证书存活市场的方法和前述计算应用更新频率的方法稍有类似，本文把某个安全证书关联的所有样本都找出来，提取出其中最早和最晚被爬取的样本的发布日期时间戳，然后将他们的差值的绝对值当作是这个安全证书的存活时长。从时长上爬取到样本的日期与样本在应用市场上实际能存活的时间稍有不同，但由于 Janus 的爬虫工具每日都从应用市场中爬取样本，而本文并没有其他方法可以知晓某款 App 具体在应用市场中上架了多久，只能近似地将上述提到的时间差当作是某个安全证书能在应用市场上存活的时间。

正如图 3.6 所示，仿冒应用安全证书存活时间的分布表明几乎所有仿冒应用安全证书都只能存活相当短的时间，这表明大多仿冒应用只会在一个很小的时间窗

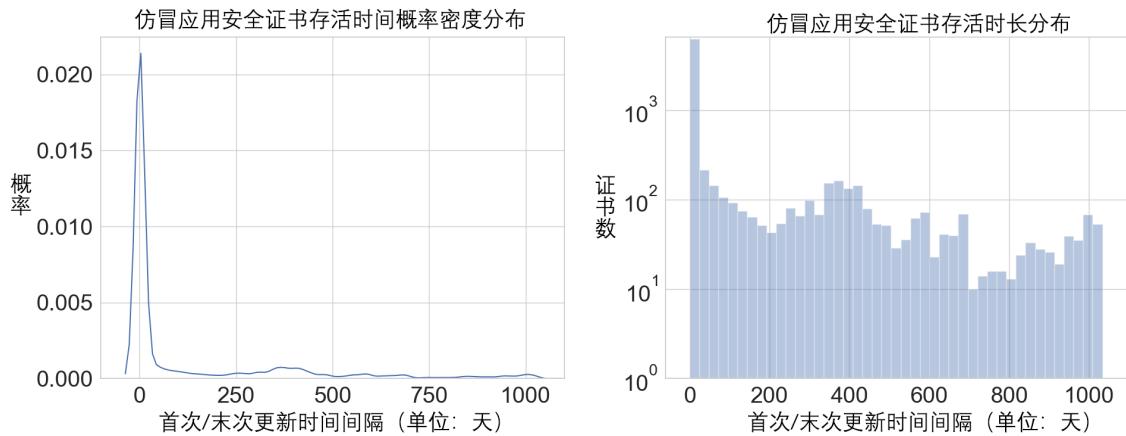


图 3.6: 仿冒应用安全证书存活时间分布

口里出现，然后迅速消失。这可以由大部分市场都有的一个安全机制解释。只要一款 App 被发现具有恶意行为或者违法行为，应用市场就会禁止开发者再使用该证书上传应用，也就是常见的封号处理。但是，也有一部分的仿冒应用安全证书存活了相当久的一段时间。根据图表信息，可以看到有的仿冒应用安全证书的生命周期甚至贯穿了本文整个研究截取的时间周期。对于这个异常样本，本文会在后续的案例分析中有更详尽的案例分析。

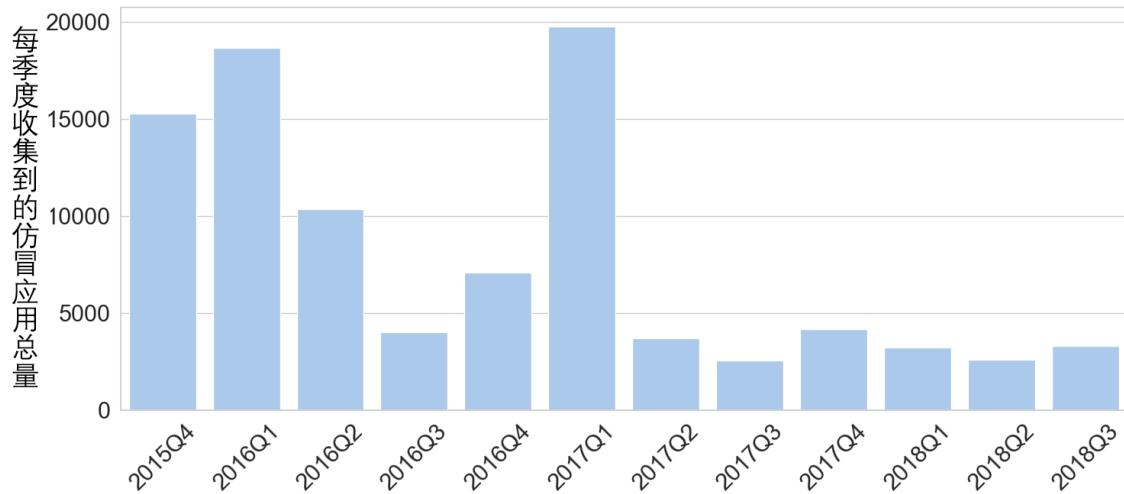


图 3.7: 每季度爬取到的仿冒样本数量（2015 年第四季度到 2018 年第三季度）

### 3.4.3 仿冒应用的行业变迁

随着时间推移，仿冒应用这个灰色产业是否有过变迁？本节对此提出了研究问题：

**RQ 3.3:** 仿冒应用是否存在一个随着时间变化的模式？

本研究统计了不同时间段所搜集到的样本数量，得出了结果如下：

**RQ 3.3.** 结果图 3.7 呈现了 Janus 自 2015 年第四季度起在每个季度爬取到的仿冒样本的总量。尽管 Janus 每个季度都能爬取到新上架的大量仿冒样本，图表信息表明，市场上仿冒应用的总数正在呈现逐年减少的趋势。要注意的是，此处的统计仅仅针对仿冒应用。因此这个现象并不表明移动黑灰产的发展具有萧条的趋势。相反地，本文认为这个现象更有可能是由黑灰产内部改革而导致的。

从一方面看，随着信息技术发展，应用市场逐渐配备了更智能、严格而又强大的监管机制和安全系统，本研究中的仿冒应用开发者无疑更难以将仿冒的应用投放到市场上了。从另一方面看，新一代的恶意软件，比如 WannaCry 等勒索软件<sup>[80]</sup> 也正在影响着整个移动黑灰产工业。与仿冒应用相比，这些新一代的恶意软件不仅更难以防范（传统的反病毒软件思路是提取已有恶意代码的特征，从而识别恶意行为，但这无法遏制新型的恶意行为），而且也似乎更容易能获取暴利。Wannacry 作为一种新型恶意软件，自 2017 年第二季度被首次发现开始，就能在数周内攻克数以千万计的设备，并让比特币的价格像搭火箭一样直线飙升<sup>[81]</sup>。之后不久，在 2018 年的第一季度，移动设备上也爆发了一系列的加密采矿恶意软件<sup>[82]</sup>。所以，本研究有理由猜测 2017 年第二季度和 2018 年第一季度的仿冒应用上架数量下跌是受到了新形态黑灰产的冲击。当然，证实这个猜想还需要采集更多数据、进行更深入的研究。

### 3.4.4 案例 3. 与大量仿冒应用相关联的仿冒应用安全证书

第 3.2 节提到有部分仿冒应用安全证书关联着多个仿冒样本。其中，一个 SHA1 码为 “61ed377e85d386a8dfee6b864bd85b0bfaa5af81” 的安全证书是所有证书中关联仿冒样本最多的，足足有 1,374 个样本持有这个仿冒证书。不仅如此，这个安全证书还是其中一个在应用市场中存活时间最长的仿冒应用安全证书。它的出现横穿

了本研究的整个研究时间过程（接近三年），而且在本研究的数据收集流程结束前依然呈现活跃状态。

最初，本研究假定这个证书属于某个通过群众分析团队验证的良性应用，因为它关联的样本数量实在是太大了，这个数量甚至超过了本研究中某些目标 App 的样本总量。然而在人工审核之后，本研究却有意料之外的结果。这个证书关联的所有 1,374 个样本都是典型的仿冒样本，其中既有只与原版应用稍微近似的模仿应用，也有外观上完全模仿原版应用的高仿应用，覆盖了本研究能找到的 47 个目标 App 中的 37 个（79%）。而这些证书关联的一些样本，甚至有自己的版本顺序，这表明有的开发者真的会追踪官方 App 的各个版本来更新、甚至维护自己的仿冒版本。

表 3.4: 由“61ed377e85d386a8dfee6b864bd85b0bfaa5af81”签署的部分样本

应用名	包名	大小
QQ Talk	net.in1.smart.qq	465.8 KB
QQ	com.h	8.2 MB
爱微信	com.lovewechat	368.4 KB
微信	com.tencent1.mm	22.1 MB
UC Mini	com.uc.browser.en	2.1 MB
UC 浏览器	com.UCMobile.microsoft	21.3 MB
Clean Master	com.blueflash.kingscleanmaster	972.0 KB
WiFi 万能钥匙	com.snda.wifilocating	5.9 MB

本文在表 3.4 中展现了由这个安全证书签署的部分样本。这批样本随后被上传到知名在线反病毒引擎 VIRUSTOTAL<sup>[79]</sup> 上。结果显示，与该证书关联的所有样本都是恶意样本（广告软件、间谍软件或木马软件等）。到目前为止，由这个证书签署的仿冒样本已经在本研究的 20 个应用来源（即应用市场）中出现，包括应用宝和 360 应用市场等主流应用市场。除此之外，百度手机助手从 2015 年起就开始接受由这个安全证书签署的应用，直到本研究的数据收集阶段结束前——数据显示，这个安全证书在 2018 年 9 月 15 日还在百度手机助手上线了一款应用。

在这里，可以得到以下两个结论：

- 就算是领先的应用市场（和顶尖的开发者）在检测恶意应用方面也不能做到尽善尽美，而现有的检测方法也有所不足，未能及时地找出可疑的开发者；
- 从这个证书在多个市场都存在的现象，本研究推导，现有的应用市场缺乏有效的信息交换机制。如果各个应用市场能建立一个互通信息的平台，分享可疑开发者/恶意开发者信息，那么将可以杜绝一部分恶意开发者在各个应用市场上到处流窜的现象。

**本节小结：**仿冒应用可以在极短时间内被研发并上架，而仿冒样本逐年下降的新增量，也许表明了仿冒应用产业正在陷入衰退期，但这需要更多证据和研究证实。此外，只有很小一部分的仿冒应用安全证书可以存活很长的一段时间，这表明应用市场的保护监管机制在一定程度上的确能发挥作用，但案例数据同时表明，现有检测方法仍有不足。另外，案例提供的数据也表示了应用市场之间缺乏交流恶意开发者/可疑开发者信息的平台。

### 3.5 本章小结

本章先分别从仿冒应用的基本特征、影响仿冒应用数量的因素和仿冒应用的发展轨迹三个不同视角对采集到的数据进行了分析，并在每个视角后的本节小结中概括了每个视角的结论，解读仿冒应用的特征。在每个视角的解读之后，本章还从数据集中选出了一些较有代表性又或者反直觉的数据样本，提供了3个不同的案例分析，在为本文的发现提供有力支持之外，也揭示了更多仿冒应用开发者的行  
为，深化了对仿冒应用生态的了解。

回看三个案例，不难发现，仿冒应用开发者的确会抓住一切可能的机会，利用包括签名机制漏洞、市场审查机制缺陷在内的各种办法制作出仿冒甚至是恶意应用。同时，本章的三个案例也说明了无论是开发人员还是应用市场，都应该在保护Android的软件安全方面上投入更大精力，从而更好地防范来自移动黑灰产的各种攻击。

在对仿冒应用进行全方位特征解读后，本文进一步进行了面向仿冒应用的排名欺检测，以探究两者之间是否存在联系。相关研究由下一章呈现。

## 第四章 面向仿冒应用的排名欺诈验证方法

上一章的研究对仿冒应用进行了全方位的特征解读。进一步地，为了解仿冒应用与另一个移动黑灰产环节—排名欺诈的关系，本文搜集了应用评论数据，并采用两种新方法以进行排查。本章呈现了相关研究的细节。

### 4.1 研究概况

应用市场中的用户反馈区是 Android 应用生态的重要部分，热心用户会在评论区提出反馈，黑灰产从业者则会利用排名欺诈的手段牟取利益。为了进一步了解仿冒应用的生态，本章针对仿冒应用与排名欺诈是否存在关联进行了研究。现有的排名欺诈检测技术各有其限制。AppWatcher<sup>[83]</sup> 提出的迭代算法需要采用已知存在排名欺诈行为的应用作为迭代起点，但真实数据的获取是本领域的公认难题；另一研究<sup>[45]</sup> 则需要持续收集应用在商店中的排名数据以进行异常检测，对收集的数据内容有一定要求。

针对上述研究存在的问题，本章研究创新性地从社交媒体研究中引进了用户行为相似度的概念，在不需先验知识和特殊数据的前提下，从用户可信度角度挖掘可能存在的排名欺诈用户群体。之后，针对评论数据量大带来的大运算量问题，本章研究又进一步采用了基于 NLP 的方法，从评论内容重复程度挖掘可能利用了排名欺诈手段的应用，实现对排名欺诈的快速排查。最后，本章对评论数据进行人工复核，证明两种方法的有效性，也确认了仿冒应用与排名欺诈相关的事。

### 4.2 仿冒评论数据收集

由于 Janus 平台上并不提供评论数据，FakeRevealer 的爬虫模块也只支持应用爬取，所以本研究另外在应用市场上重新收集了评论数据。在 360 手机助手应用商店中，本研究随机挑选了 856 个应用，爬取了这些应用的 APK 包和对应的所

有历史评论。之后，本研究将上一次数据收集时保存的正版应用的信息重新导入 FakeRevealer，从新收集的应用中筛选出了对应仿冒应用。

每条评论都会附带一个评价分数，某款 App 在市场上的平均评价就是所有评论评分的均值。对于每条评论信息，本研究能收集到的数据项是：应用包名、用户 ID、评论内容、评分和评论日期共五项。

#### 4.2.1 评论数据概览

在本研究收集评论的所有 856 个应用中，有 6 款应用与先前仿冒应用数据中的包名对应。要注意的是，由于本研究的仿冒应用列表为针对 50 个热度最高的目标应用整理而成，而收集评论的应用是在整个市场范围内随机挑选的，所以这里的仿冒应用占总体应用比例较小。但这不意味着整个市场中就只有这几款应用是仿冒应用。另外，由于这 856 个应用是随机挑选的，本研究认为这批数据具有一定的代表性，可以应以反映整个市场的评论分布情况。本次研究一共爬取到了 267,397 名用户的 365,461 条评论，其中 6 款仿冒应用的所有历史评论共计 3,591 条，来自 2,946 名用户。

#### 4.2.2 基础分析

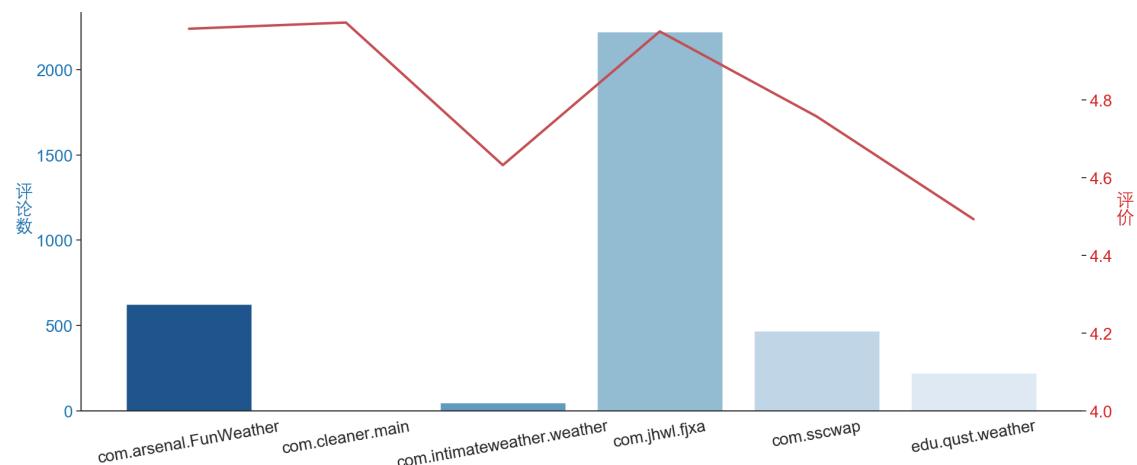


图 4.1: 各仿冒应用在 360 手机助手商店中的评论数量与评分分布

图 4.1 显示了 6 款应用收到的评论数量和评分分布。蓝色的柱状图表示评论数量，红色的折线图表示各评论汇总后的平均评分（以 5 分为满分算）， $x$  轴分别代

表不同样本的包名。按评论数从多到少的顺序看，*com.jhwlfjxa* 收到了 2223 条评论，平均评价为 4.98 分；*com.arsenal.FunWeather* 收到了 626 条评论，平均评价也是 4.98 分；*com.sscwap* 收到了 467 条评论，平均评价 4.76 分；*edu.qust.weather* 收到了 223 条评论，平均评价 4.49；*com.intimateweather.weather* 和 *com.cleaner.main* 分别只收到了 49 条和 3 条评论，而他们的平均评价则分别为 4.63 分和 5 分。乍眼一看，上述应用的评分都十分高。以下是一些热门应用的评分和这些仿冒应用的评分的对比：在同一市场下，移动购物类应用淘宝的评分为 4.55 分，近年十分受欢迎的短视频应用抖音平均评价是 4.5 分，游戏类应用开心消消乐的评分是 3.65 分，而微信的平均评价更是只有 3.45 分。上述应用毫无疑问都是十分优质的 App，庞大的用户基数带来的大量真实评价会使得平均评价较为稳定，不会因为在短时间内收到少量好评或者差评就产生较大的评价波动。在淘宝等应用作为基准的情况下，6 款仿冒应用的评分之高不禁令人联想到恶意刷评的相关研究。然而，本研究不能仅凭几个应用的平均评价对比就咬定仿冒应用存在刷好评的行为。因此，本文先分析数据集的整体分布，再作进一步比较。

在不考虑上述提到的恶意刷评的情况下，一款应用的使用人数越多，就越有可能收到来自用户的评价。所以可以在一定程度上，从一款应用的评论数目估计其用户数量的多少。图 4.2 中的两个小提琴图显示了所有 856 个应用收到评论的数量和总体评级分布，有助于了解市场中应用的热度分布情况和用户的评价倾向。

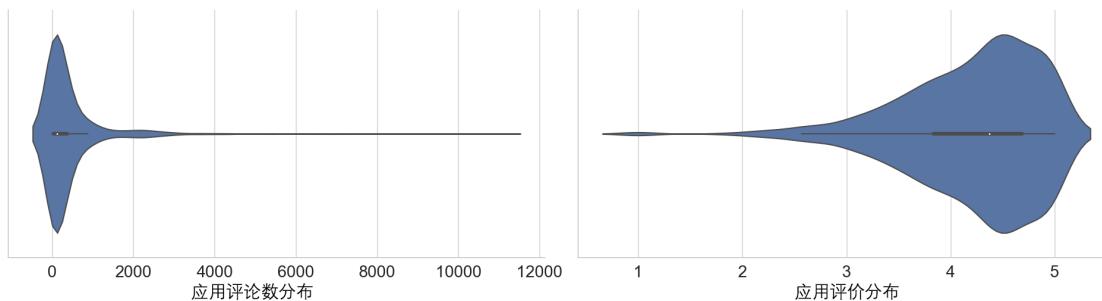


图 4.2: 所有 856 个应用在商店中的评论数量与评分分布

左边的小提琴图表示每个应用收到的评论总数分布，其四分位数分别为 31, 124 和 375.75。这说明，在收集到的 856 个应用中，25% 的应用收到小于或者等于 31 条评论，50% 的应用收到小于或等于 124 条评论。如果某款应用收到的评

论数大于 375 条，那这款应用的评论总数就能排在前 25% 了。另外，数据显示，仅有 5% 的应用收到了超过 2,109 条评论。结合仿冒样本的数据，*com.jhwl.fjxa*、*com.arsenal.FunWeather* 和 *com.sscwap* 的评论数量都排在了前 25%，*com.jhwl.fjxa* 更是能排在前 5% 的位置。从数据上看，上面三款 App 相当受欢迎。

右边的小提琴图则表示各个应用收到的平均评价的分布情况，其四分位数分别为 3.84, 4.37 和 4.69，约 5% 的应用平均评价为 5 分满分。这个分布说明这个应用市场上的用户十分倾向于给出高分评价，至少有过半数的评论都是满分好评。回到仿冒样本的数据，其中有两款评论少于 100 条，很可能存在较大的个体偏差，在此先忽略不计。余下四款仿冒应用中也有三款的平均评价排进了前 25%，恰好也是评论数较多的 *com.jhwl.fjxa*、*com.arsenal.FunWeather* 和 *com.sscwap*。

结合两个维度的分布结果，*com.jhwl.fjxa*、*com.arsenal.FunWeather*、*com.sscwap* 三款 App 不仅用户众多，而且好评如潮。再回望淘宝、微信等应用的评分，两者之间似乎有了矛盾。一款应用的用户越多，真实的评论数目越大，该款应用的评分就会趋向客观，直到收敛到一个可以反映应用质量的真实水平。*com.jhwl.fjxa*、*com.arsenal.FunWeather*、*com.sscwap* 三款应用的用户量固然不能和微信、淘宝相比，但成百上千的评论数也暗示着一个不小的用户群体。在用户群体有一定规模的情况下，还能保持高水平的平均评价不容易，因此本文假设上述应用利用了排名欺诈手段将评分拉高，并将在后续章节对此假设进行验证。

## 4.3 仿冒应用与排名欺诈关联验证

### 4.3.1 排名欺诈检测初探

由于前文提及的两项排名欺诈检测研究分别需要先验知识与特殊数据，而现有数据中并不具备这些条件，所以本研究先选择了另一前人研究的方法进行探索。**FRAUDAR**<sup>[50]</sup> 是 Bryan Hooi 在 2016 年提出的算法，使用二分图挖掘的方法找出可能的虚假好评，并输出最可能涉及排名欺诈的应用和用户。算法基于的假设是，普通用户的行为（在本文中即为对 App 评论的行为）是大致随机的，而用于进行排名欺诈的用户群的行为却会有比较明显的指向性（即针对购买了排名欺诈服务的 App 发送好评），而且为了将平均评分拉高，就意味着需要发送大量好评。如果将

用户和 App 分别看做两种不同节点，每条好评看作是两种节点之间的边，那么在这张用户-评论-应用图中，排名欺诈用户和对应的 App 之间就会有特别紧密的联系。如果能把这个联系特别紧密的子图找到，就有可能从中找到真实的排名欺诈用户和应用。

由于此处的排名欺诈指用大量虚假好评刷高应用的平均评价，所以在寻找排名欺诈用户和对应应用时，应该只采用满分好评作为数据。因此，本研究从所有评论中筛选出了满分好评，其总数为 381,507 条，由 229,100 名用户给出，分布于 848 个应用中，占评论总数的 87.15%。本研究将 FRAUDAR 应用在了本研究的好评数据集上，找到了 115 名可疑用户和 13 个可疑应用。经过比对后，本研究发现，13 个可疑应用并不包含仿冒样本，而仿冒应用的所有评论条目中也没有源于那 115 名可疑用户的评论。FRAUDAR 的工作原理是在二分图中不断删边，从而获得一个紧密子图，这意味着可能潜在的大量假阴性结果（False Positive）。在不确定 FRAUDAR 有效性的情况下，本研究提出了另一种排查方法与之进行对比。

#### 4.3.2 基于评论用户可信度的排名欺诈排查

Mohammad-Ali<sup>[84]</sup> 在 2013 年提出了一个计算社交媒体中用户行为相似度的计算方法。评论区虽然不同于社交媒体，但两者之间有一些共同之处，可以尝试将该算法迁移至排名欺诈检测中。在该算法的基础上，本研究创新性地加入用户可信度权重的概念，基于用户行为相似度对用户进行聚类，最终找出可疑用户群体，从而避免 FRAUDAR 的高假阴性结果。

该算法以式 4.1 计算用户行为相似度，如果相似度超过了某个阈值  $T_1$ ，就可以将两名用户聚入同一类。式 4.1 中的  $B(u_i, t)$  指用户  $u_i$  在时间节点  $t$  的行为（在此处可以理解为对某一个应用给好评），而  $\sigma(B(u_i, t), B(u_j, t))$  则是一个用来计算用户  $u_i$  和  $u_j$  在时间为  $t$  时行为相似度的方程，Mohammad 在文中选用的是式 4.2 所示的 Jaccard 相似系数，所以本文在这里也选用了同样的 Jaccard 系数计算。

$$Sim(u_i, u_j) = \frac{1}{t_n - t_0} \sum_{t=t_0}^{t_n} \sigma(B(u_i, t), B(u_j, t)) \quad (4.1)$$

$$Jaccard(set_i, set_j) = \frac{|set_i \cap set_j|}{|set_i \cup set_j|} \quad (4.2)$$

在这种计算方式下，那些仅给过一次好评的用户将会很容易成为噪声数据，对研究的结果产生影响，所以本研究剔除掉了这部分数据。仅给过一次好评的用户共 186,775 人，占所有给出好评用户的 81.53%。

在将行为相似的用户聚类之后，本研究引入用户可信度权重。可以通过计算某一应用评论中疑似排名欺诈评论的占比、或是评论该 App 的可疑用户占所有可疑用户的占比来排查可能购买了排名欺诈服务的 App。

#### 定义 4 应用的用户可信度权重

假设所有市场用户的集合为  $G_{all}$ ，已知疑似排名欺诈用户群体  $G_r$ ，用户以  $u$  表示，由任意用户  $u_i$  发布的评论  $cmt_j$  表示为  $u_i \rightarrow cmt_j$ ，市场中的某一应用  $app_k$  的评论列表为  $CL_{app_k}$ 。则该应用  $app_k$  的用户可信度权重  $W_{app_k}$  可由式 4.3 中的二元组表示：

$$W_{app_k} = (w_{app_k}^0, w_{app_k}^1) \quad (4.3)$$

$$w_{app_k}^0 = \frac{|\{u_i \mid u_i \in G_r, cmt_j \rightarrow u_i, cmt_j \in CL_{app_k}\}|}{|G_r|} \quad (4.4)$$

$$w_{app_k}^1 = \frac{|\{cmt_j \mid cmt_j \in CL_{app_k}, cmt_j \rightarrow u_i, u_i \in G_r\}|}{|CL_{app_k}|} \quad (4.5)$$

在计算完权重之后，可以分别按其中的两个子权重对应用进行排名，筛选出可能购买了排名欺诈服务的 App。

本研究分别将  $T_1$  设置为 0.4, 0.6 和 0.8，尝试对可疑用户进行聚类，结果分别将 42,325 名给出好评次数大于 1 的用户分到了 10,024, 14,520, 15,493 个聚类中。可对这些聚类进行简单分析如下：绝大多数聚类中都只有一名用户，即使是在相似度阈值只有 0.4 的情况下，也只有约 7% 的聚类中包含 3 个或以上的用户（阈值为 0.6 和 0.8 时，该比例均为 6%）。但是，当挑出包含用户数目大于 10 的聚类 ( $T_1$  为 0.4/0.6/0.8 时，这些聚类的占比分别为 0.97%/0.70%/0.57%) 时，这

些聚类却分别包含了 29,168/23,742/22,218 个用户，他们所发布的好评共计分别有 98,226/80,492/73,422 条。由此可推定，有一部分用户的行为模式相当近似且可疑，本文将会把这部分用户组成的群体看作是疑似的排名欺诈用户群体 ( $G_r$ )。

接下来，本研究分别计算了不同  $T_1$  下，三款仿冒应用在式 4.4 和式 4.5 中的两个权重，即三款应用中的好评用户占可疑用户的比例、以及其好评占所有可疑用户发布的好评的比例。对于本章研究的三个仿冒应用，其两个子权重的结果分别展示在表 4.1 和表 4.2 中。

表 4.1: 各应用用户可信度权重及对应排名 (一)

包名	$w^0(T_1=0.4)$	排名	$w^0(T_1=0.6)$	排名	$w^0(T_1=0.8)$	排名
com.arsenal.FunWeather	0.97	2	0.9	6	0.82	9
com.jhwl.fjxa	0.85	15	0.84	12	0.8	12
com.sscswap	0.02	303	$5 \times 10^{-3}$	346	$5 \times 10^{-3}$	318

表 4.2: 各应用用户可信度权重及对应排名 (二)

包名	$w^1(T_1=0.4)$	排名	$w^1(T_1=0.6)$	排名	$w^1(T_1=0.8)$	排名
com.jhwl.fjxa	0.05	11	0.06	10	0.07	11
com.arsenal.FunWeather	0.02	45	0.02	39	0.02	39
com.sscswap	$2 \times 10^{-4}$	261	$8 \times 10^{-5}$	279	$9 \times 10^{-5}$	251

结果显示，无论是用哪种权重对应用可疑度进行排名，*com.arsenal.FunWeather* 和 *com.jhwl.fjxa* 在总计的 848 个应用中都排在相当靠前的位置，所以这两个应用都相当可疑，十分可能具有排名欺诈行为。另一方面，*com.sscswap* 的排名相对靠后，具有排名欺诈行为的可能性较小。

本组实验使用了服务器承担运算任务，实验服务器搭载了两颗 Intel 的至强 E5-2367 V4 版 8 核 CPU，内存为 252GB。在  $T_1$  分别设置为 0.4/0.6/0.8 时，三组基于用户可信度实验用的 python 代码分别需要运行 7,086/6,935/6,801 分钟得到出结果。

### 4.3.3 基于评论内容相似度的排名欺诈排查

上节的方法带来了相对可信的结果，但同时也导致了庞大的运算任务。为了减小运算量，提高排查速度，本研究提出了利用评论内容相似度排查排名欺诈的方法。根据经验，在排名欺诈相关的评论通常具有很高的相似性，甚至一模一样，导致那些购买排名欺诈服务的应用中有很多相似甚至相同的评论。所以，可以通过计算应用内相似评论的比率以筛选可能购买了排名欺诈服务的应用。

#### 定义 5 应用评论重合率

对于市场中的某一个评论列表为  $CL_{app_k}$  的应用  $app_k$ ，假设其所有评论可以被分成  $n$  个组  $CG_i(0 < i < n)$ ，则定义该应用的评论重合率  $RD_{app_k}$  如下。重合率越高，应用越有可能存在排名欺诈行为。

$$RD_{app_k} = 1 - \frac{\sum_{i=0}^n |CG_i|}{|CL_{app_k}|} \quad (4.6)$$

为了找出内容高度重合的评论，研究者可以用 NLP 中的词袋模型（Bag-of-words Model）将每个评论转化成一串词语列表。具体做法是，先对每条评论进行分词，形成词袋，再用一种合适的标准去衡量不同词袋之间的相似度，并将相似内容聚为一类。分词方面，本研究使用了中文分词项目“结巴”中文分词<sup>1</sup>，为了更好地从词袋中提取语义信息，本文还从网上整理了一份停用词（Stop words）表，在分词之后筛去停用词，以减少不含语义的停用词对相似度计算造成的干扰。词频太低的词语也可能会影响聚类产生影响，为此，本研究会在聚类前从各个词袋中删除总词频太低的词语。而在衡量词袋相似度方面，本文再次使用了式 4.2 的 Jaccard 相关系数。

另外，本研究要筛查的是排名欺诈行为，其本质是通过提供大量虚假好评提高应用的平均评价，所以还要从数据集中除去一部分评论较少的应用，因为他们不太可能购买了排名欺诈服务。本文分别从数据集中剔去了总评论少于 50 条和总评论少于 100 条的应用，使得数据组中分别剩下 511 和 455 个应用参与排名。

<sup>1</sup><https://github.com/fxsjy/jieba>

在预处理过程中，本研究剔除了在所有评论中出现次数小于等于 2 的词语，然后以 0.8 为相似度阈值对评论进行聚类。

结果可见图 4.3，其中图例上标注的“50”和“100”分别表示剔除了总评论少于 50 条的应用的数据组和剔除了总评论少于 50 条的应用的数据组，两个图形中的三条虚线分别是两组数据中的 3 个四分位数线。“50”数据组的三条四分位数线分别对应  $x$  轴上 0.09, 0.17 和 0.30 的位置，说明数据组中有 25% 的应用评论重合率小于 9%，50% 应用的评论重合率小于 17%，如果某应用的评论重合率大于 30%，那么该应用的评论重合率就排在数据集的前 25% 了。与之类似，“100”数据组的三条四分位数线分别对应  $x$  轴上 0.10, 0.18 和 0.32 的位置，表明两组数据整体的评论重合率并不高。此外，“100”组的数据分布比“50”组的数据分布稍微偏向  $x$  轴右侧，证明接收到评论较少的应用的评论重合率的确也偏低。

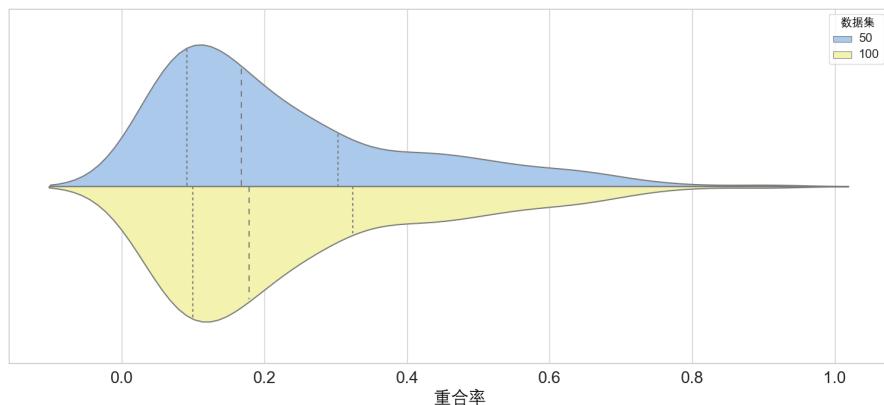


图 4.3: 两种数据组的应用评论重合率结果

表 4.3: 评论重合率结果

包名	评论重合率 (%)	排名 (数据组 “50”)	排名 (数据组 “100”)
com.arsenal.FunWeather	47.11	58	57
com.jhwl.fjxa	44.97	66	65
com.sscswap	10.98	349	325

表 4.3 提供了三款仿冒应用的评论重合率和在两组数据中的排名情况。在“50”数据组的 511 个应用中，*com.arsenal.FunWeather* 和 *com.jhwl.fjxa* 分别排名 58 和 66

(前 11% 和前 13%)，而在“100”数据组的 455 个应用中，*com.arsenal.FunWeather* 和 *com.jhwl.fjxa* 的排名是 57 和 65 (前 13% 和前 14%)，均十分靠前。而且，只看评论重合率，两款应用的数值都超过了 44%，相当于约每五条评论中就有两条十分类似的评论，这表明上述两款应用很有可能使用了排名欺诈。*com.sscswap* 相对靠后的排名和较低的重合率说明其好评比较多元化，使用排名欺诈服务的可能性比较低。上述结果与基于评论用户可信度的排名欺诈排查方法的结果相对一致，说明两种方法效果相当。由于本方法的运算只需用到单个应用的所有应用数据，所以在只有单个应用的评论数据可用时，也能使用本方法进行排查。

#### 4.3.4 人工复核

最后，本研究对 *com.arsenal.FunWeather*、*com.jhwl.fjxa* 和 *com.sscswap* 的评论进行了人工复核，验证上述三种方法得到的结果。为了方便复核，本研究对每个应用的评论都按内容进行了排序。

图 4.4 展示了本文对三款应用好评取样的结果。可以看出，*com.jhwl.fjxa* 和 *com.arsenal.FunWeather* 两款 App 都有明显的评论重复现象。而图中对 *com.sscswap* 的评论虽然看上去也比较近似，但这其实是由于本文按照评论内容对这些评论进行了一次排序。如果结合日期数据观察，就能发现这些数据是由用户在比较分散的时间发出的。所以说这些看上去稍微类似的评论，并不一定存在关联关系，本研究不倾向于认为这款 App 购买了排名欺诈服务。反观 4.4a 中的评论，有多条长评论高度相似、甚至一模一样，这在真实案例中是不太可能会存在的情况；而 4.4b 中多条近似、相同的评论是在十分接近的时间里被发表的，进一步加深了他们之间存在关联的可能性。

综上，本研究有理由相信这两款仿冒应用的确存在排名欺诈的行为，从而推导仿冒应用与排名欺诈存在关联。上述结果也同时说明了本研究提出的两种方法都能有效鉴别出使用了排名欺诈的应用，效果优于现有的部分研究：如前述的 FRAUDAR 只能从本研究的评论数据集中发现 15 个可疑应用，但根据上一节的实验结果，至少有 60 个应用很可能存在排名欺诈行为，而本研究中提出的两种验证方法均可将可疑应用有效地筛选出。

ID	包名	评论	用户	日期	评级
2360862	com.jhwl.fjxa	为了找对象才下载这个软件的	迎蓉元香51a	2017/10/14	5
2360084	com.jhwl.fjxa	为了找对象才下载这个软件的	360U1631348866	2017/12/26	5
2361916	com.jhwl.fjxa	为了追女朋友才下载的，现在已经追到手了	用铁	2017/5/15	5
2361386	com.jhwl.fjxa	为了追女朋友才下载的，现在已经追到手了	人生时间是一把	2017/8/8	5
2360732	com.jhwl.fjxa	主动的创造自己的事业，同样要主动争取自己的爱情	360U1355330743	2017/10/26	5
2361893	com.jhwl.fjxa	主要是实名制的，很棒	花如你	2017/5/17	5
2361363	com.jhwl.fjxa	主要是实名制的，很棒	点点沥	2017/8/9	5
2360685	com.jhwl.fjxa	之前也是听同事推荐注册了一个号码在里面泡了一阵子，自己感觉还不错	360U376135101	2017/10/31	5
2360603	com.jhwl.fjxa	之前也是听同事推荐注册了一个号码，自己感觉还不错	AnkeSweet	2017/11/10	5
2361819	com.jhwl.fjxa	之前我附近的人，我都不认识他们，下载这个软件之后就熟了好多了	chuoyan12733	2017/5/23	5
2361603	com.jhwl.fjxa	之前我附近的人，我都不认识他们，下载这个软件之后就熟了好多了	时就像	2017/6/16	5
2361295	com.jhwl.fjxa	之前我附近的人，我都不认识他们，下载这个软件之后就熟了好多了	悲伤了别	2017/8/16	5
2360668	com.jhwl.fjxa	之前我附近的人，我都不认识他们，下载这个软件之后就熟了好多了	雪梦玉	2017/11/4	5
2360315	com.jhwl.fjxa	之前我附近的人，我都不认识他们，下载这个软件之后就熟了好多了	藏羚羊1234	2017/12/9	5
2360557	com.jhwl.fjxa	也没报什么太大的希望，就是想找人聊天。	船长在忙	2017/11/14	5
2360455	com.jhwl.fjxa	也没报什么太大的希望，就是想找人聊天。	360U296426264	2017/11/24	5
2360940	com.jhwl.fjxa	交友上面的人很靠谱，感觉不错	无奈地躲在属	2017/10/5	5
2360937	com.jhwl.fjxa	交友上面的人很靠谱，感觉不错	知道岁	2017/10/5	5

(a) com.jhwl.fjxa 部分评论

ID	包名	评论	用户	日期	评级
4193186	com.arsenal.FunWeather	不容错过的软件！	HAPPENF	2015/9/23	5
4193185	com.arsenal.FunWeather	不容错过的软件！	正义微博	2015/9/23	5
4193020	com.arsenal.FunWeather	不容错过的软件！	乱了思绪邢	2015/10/16	5
4193034	com.arsenal.FunWeather	不容错过的软件！	无可取代热幌	2015/10/16	5
4193009	com.arsenal.FunWeather	不错。赞一个	浅尝辄止仄懒盏	2015/10/16	5
4193330	com.arsenal.FunWeather	不错不错。你可以试试看	情渊配圣鑑	2015/9/22	5
4193331	com.arsenal.FunWeather	不错不错。你可以试试看	艾薇儿带我装逼	2015/9/22	5
4192976	com.arsenal.FunWeather	不错不错。你可以试试看	物是人非幼脸僚	2015/10/16	5
4192804	com.arsenal.FunWeather	不错不错。你可以试试看	冉渤严	2015/10/19	5
4192791	com.arsenal.FunWeather	不错不错。你可以试试看	安之若素九坷盏	2015/10/19	5
4193130	com.arsenal.FunWeather	不错值得推荐	打尽天下的酱油	2015/9/23	5
4193129	com.arsenal.FunWeather	不错值得推荐	药大老表	2015/9/23	5
4193048	com.arsenal.FunWeather	不错值得推荐	几番轮回壮	2015/10/15	5
4192831	com.arsenal.FunWeather	不错喔，真实个暖人的软件	容颜殆尽乱略	2015/10/18	5
4192965	com.arsenal.FunWeather	不错好用。	花落半歌汕优	2015/10/16	5
4192820	com.arsenal.FunWeather	不错好用。	森林散布截	2015/10/18	5
4192951	com.arsenal.FunWeather	不错实用。。	孤峰无伴见撑	2015/10/17	5
4193168	com.arsenal.FunWeather	不错很好很强大	YoseO晕	2015/9/23	5

(b) com.arsenal.FunWeather 部分评论

ID	包名	评论	用户	日期	评级
3134829	com.sscwap	不错 希望部分改进更完美	飞一样流年	2013/7/22	5
3134595	com.sscwap	不错 温度风力都有 赞！	力图蘑菇	2014/3/1	5
3134611	com.sscwap	不错。占用内存小。主要是有附近的县市。不之有地区市。	瑞士银行001	2014/2/26	5
3134656	com.sscwap	不错不错，很简单。	360U123170745	2014/2/3	5
3134735	com.sscwap	不错啊	疯子爵士	2013/12/14	5
3134635	com.sscwap	不错的，非常好	360U679162839	2014/2/13	5
3134874	com.sscwap	不错！一直再用！软件体力小省流量主要是没有广告！支持作者	悲伤的初吻	2013/3/22	5
3134822	com.sscwap	不错！省空间，就是有时表慢，期待更好！	renz0906	2013/7/29	5
3134744	com.sscwap	不错！赞一下！	360U319733778	2013/12/11	5
3134504	com.sscwap	不错，但是桌面插件到哪儿去了？	亲爱的V用户	2014/6/12	5
3134654	com.sscwap	不错，但有待改进	小兰海豚	2014/2/4	5
3134512	com.sscwap	不错，值得推荐！	djx001144	2014/4/30	5
3134858	com.sscwap	不错，实用	拖hi宇	2013/4/10	5
3134612	com.sscwap	不错，干净，没有广告，不影响系统运行。	360U286826485	2014/2/25	5
3134763	com.sscwap	不错，挺好的~	spider_yang	2013/11/15	5
3134605	com.sscwap	不错，桌面怎么只显示前天晚上的天气?其它挺好。	苦乐笨牛	2014/2/27	5
3134785	com.sscwap	不错，简单就好，建议减去六小时提示音。	cucicg	2013/10/19	5
3134807	com.sscwap	不错，顶起！	用户273507680	2013/9/14	5

(c) com.sscwap 部分评论

图 4.4: 应用评论取样

本研究还随机抽取了一些发布相同好评的不同用户进行追查，结果发现部分用户在研究收集到的数据集中的评论数只有一到两条，这很有可能是提供排名欺诈服务的商家规避检测的一种策略。

#### 4.4 本章小结

本章从仿冒应用在市场中收到的评论入手，验证了仿冒应用与排名欺诈是否存在关联。为完成本研究，本文随机选取了部分应用的评论数据进行分析，借鉴了前人研究提出了研究手段进行筛查。

针对现有研究存在的不足之处，本研究尝试将其他领域的研究方法迁移应用于本次排查中，获得了良好的结果；为了进一步优化排查方法，减小运算量，本研究进一步提出了可用于单个应用的排名欺诈排查的方法。最后的人工复核证明，无论是基于评论内容相似度的排名欺诈排查方法还是基于用户可信度权重的排查方法，都具有有效性，且效果优于现有方法。

## 第五章 总结与展望

### 5.1 总结

在本文中，“仿冒应用”这个概念被率先引入。然后，本文对这一方面进行了专门的研究，还搜集了大量的相关样本以辅助调查。本文的前期调研结果显示，本课题是第一个针对仿冒应用进行大规模全面实证研究的课题。

为了更好地了解这个类型的应用的生态环境，本文基于 Python 3 设计实现了仿冒应用收集框架 FakeRevealer，利用基于 BFS 的算法收集了来自现实世界中各个应用市场的近 14 万个应用样本，并且从多个不同维度，对这个数据库里面的仿冒样本进行了观测和考察。这些维度包括了 APK 包中的安全证书信息、应用大小、应用名、包名和时间因素等等。

然后，本工作将收集到的数据分为了仿冒应用的基本特征、影响仿冒应用数量的因素和仿冒应用的发展轨迹三个不同视角进行了测量，获得如仿冒应用的命名倾向和仿冒应用开发者对市场监管防御机制的规避策略等信息。为了佐证本文的发现，本研究在每个视角解读之后给出了从数据集中挑选的几个研究案例，呈现了如仿冒应用开发者对不同热门应用的仿冒方式的内容。这几个案例进一步深化了本文对仿冒应用生态系统的发现。

之后，本文还收集了部分仿冒样本在商场上对应的评论和评级，排查仿冒应用是否利用了排名欺诈。由于现有的排名欺诈检测手段尚有不足，本工作创新性地分别利用两种创新的研究方法—基于用户行为的用户可信度验证和基于 NLP 的评论内容相似度验证，对数据中的排名欺诈行为进行了排查。结果显示，刷好评的排名欺诈行为的确存在于应用市场中，在本工作搜集到的仿冒应用评论中就有刷好评的痕迹。

## 5.2 展望

在大规模分析的部分中，本文中用到了三个不同的角度分别探索仿冒应用的特征，但回顾探索过程，一些方法和步骤依然不够深入。如果能从以下三个角度再向仿冒应用入手研究，或许能有更多有所裨益的发现：

- **应用图标:** 本文在进行案例研究中发现，不少仿冒应用的图标和原版官方应用的图标其实十分相像。因此，图标也可以是一个用于发掘/鉴别仿冒应用的突破口，研究者也许可以从应用图标中挖掘到更多可用的信息与行为模式。碍于时间因素所限，本文研究中并未加入图像对比处理部分提取各 APK 包中的图标与官方应用的图标进行比对，但如果能研究出快速比对多个应用间图标、图像相似度的算法，定当对应用市场的安全监管筛选机制有所好处。
- **应用内代码/文本/链接/ip 分析:** 代码分析可以有效地剖析应用的行为，而相似的文本资源、链接等信息也可以提供各个 App 之间可能存在的关联关系。遗憾的是，从当前技术水平出发，仔细地对一个 App 进行完整而全面的静态分析所需时长太长，而动态分析需要测试样例驱动，自动化的动态测试工具往往未能深入拓展一个应用的大部分核心功能。因此，开发出快速的分析算法对 App 进行更深入的探索，就能挖掘出有关仿冒应用生态的更多信息。
- **仿冒应用总量的变化原因:** 第三章中提及到了 Janus 收集到的仿冒应用数量并非一直保持上升趋势。近年来，能搜集到的仿冒应用数量有突然下跌、甚至渐次式微的迹象。究竟是什么因素导致了这个原因？是移动黑灰产内部的变化，还是安全厂商日益紧密的封锁？这将会是一个十分有趣的课题。

而在评论分析的部分中，也有可以继续发掘的部分。在现阶段，学界关于排名欺诈的研究一直在针对积极评价方面，但是对应用差评进行排名欺诈的相关研究却有待补充。刷好评可以提高应用评价提升应用排名，如果反其道而行之，用差评对目标应用进行攻击，其实也可以降低目标应用的评价，对其排名进行打击。另外，在实际上，用户给的差评中含有相当多的有用信息。用户对应用的不满、功能上的建议、bug 的反馈，都可以反映在差评上。关于用户差评，还有很多的研究空间。

最后, FakeRevealer 框架本身, 无论是代码层面还是设计层面, 也有值得改善的地方。比如是否能结合自动化爬虫框架提高爬虫模块的鲁棒性(对抗应用商店的反爬虫技术、下载稳定性), 工具本身的代码优化, 还有工具整体的易用性、稳定性等。

总之, 从整体上看, 本文的工作还有很多可以深化的部分。然而, 诚希望本文研究的结果能够为移动安全产业的从业人员(不论是工业界或学术界)提供足够的信息, 以改善移动安全界的现状; 或是抛砖引玉, 在让读者对移动应用黑色产业有更多认识的同时, 激发读者对仿冒应用等方面的研究兴趣, 并从上述几点出发, 为后人带来更多深入而完善的相关研究。

## 参考文献

- [1] STATCOUNTER. Mobile Operating System Market Share Worldwide, 2009 - 2020[EB/OL]. 2019 [February 23, 2020].  
<https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2009-2020>.
- [2] CLEMENT J. Number of available applications in the Google Play Store from December 2009 to December 2019[EB/OL]. 2019 [January 4, 2020].  
<https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>.
- [3] WASSERMAN A I. Software engineering issues for mobile application development[C] // Proceedings of the FSE/SDP workshop on Future of software engineering research. 2010 : 397 – 400.
- [4] CHEN S, FAN L, CHEN C, et al. StoryDroid: Automated Generation of Storyboard for Android Apps[C] // Proceedings of the 41th ACM/IEEE International Conference on Software Engineering, ICSE 2019. 2019.
- [5] ANDOW B, NADKARNI A, BASSETT B, et al. A Study of Grayware on Google Play[J]. 2016 IEEE Security and Privacy Workshops (SPW), 2016 : 224 – 233.
- [6] LUO L, FU Y, WU D, et al. Repackage-proofing android apps[C] // 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). 2016 : 550 – 561.
- [7] ZHOU Y, JIANG X. Dissecting Android Malware: Characterization and Evolution[J]. 2012 IEEE Symposium on Security and Privacy, 2012 : 95 – 109.
- [8] KHANMOHAMMADI K, EBRAHIMI N, HAMOU-LHADJ A, et al. Empirical study of android repackaged applications[J]. Empirical Software Engineering, 2019, 24(6) : 3587 – 3629.

- [9] ZHOU W, ZHOU Y, JIANG X, et al. Detecting repackaged smartphone applications in third-party android marketplaces[C] // CODASPY. 2012.
- [10] ZHENG M, SUN M, LUI J C S. DroidAnalytics : A Signature Based Analytic System to Collect , Extract , Analyze and Associate Android[C] // . 2013.
- [11] CRUSSELL J, GIBLER C, CHEN H. Attack of the clones: Detecting cloned applications on Android markets[C] // European Symposium on Research in Computer Security. 2012 : 37 – 54.
- [12] CRUSSELL J, GIBLER C, CHEN H. Scalable semantics-based detection of similar Android applications[C] // Proc. of ESORICS : Vol 13. 2013.
- [13] CHEN K, LIU P, ZHANG Y. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets[C] // Proceedings of the 36th International Conference on Software Engineering. 2014 : 175 – 186.
- [14] HANNA S, HUANG L, WU E, et al. Juxtapp: A scalable system for detecting code reuse among android applications[C] // International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. 2012 : 62 – 81.
- [15] LINARES-VÁSQUEZ M, HOLTZHAUER A, POSHYVANYK D. On automatically detecting similar Android apps[C] // Program Comprehension (ICPC), 2016 IEEE 24th International Conference on. 2016 : 1 – 10.
- [16] WU X, ZHANG D, SU X, et al. Detect repackaged android application based on http traffic similarity[J]. Security and Communication Networks, 2015, 8(13): 2257 – 2266.
- [17] ZHANG F, HUANG H, ZHU S, et al. ViewDroid: towards obfuscation-resilient mobile application repackaging detection[C] // WISEC. 2014.
- [18] SUN M, LI M, LUI J C. DroidEagle: seamless detection of visually similar Android apps[C] // Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks. 2015 : 1 – 12.
- [19] KYWE S M, LI Y, DENG R H, et al. Detecting camouflaged applications on mo-

- bile application markets[C] // International Conference on Information Security and Cryptology. 2014 : 241 – 254.
- [20] SOH C, TAN H B K, ARNATOVICH Y L, et al. Detecting clones in android applications through analyzing user interfaces[C] // 2015 IEEE 23rd international conference on program comprehension. 2015 : 163 – 173.
- [21] GLANZ L, AMANN S, EICHLBERG M, et al. CodeMatch: obfuscation won't conceal your repackaged app[C] // Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017 : 638 – 648.
- [22] WANG H, GUO Y, MA Z, et al. WuKong: a scalable and accurate two-phase approach to Android app clone detection[C] // Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015 : 71 – 82.
- [23] ZHOU W, ZHANG X, JIANG X. AppInk: watermarking android apps for repackaging deterrence[C] // Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. 2013 : 1 – 12.
- [24] ZHOU W, WANG Z, ZHOU Y, et al. Divilar: Diversifying intermediate language for anti-repackaging on android platform[C] // Proceedings of the 4th ACM conference on Data and application security and privacy. 2014 : 199 – 210.
- [25] ZENG Q, LUO L, QIAN Z, et al. Resilient decentralized Android application repackaging detection using logic bombs[C] // Proceedings of the 2018 International Symposium on Code Generation and Optimization. 2018 : 50 – 61.
- [26] LI L, GAO J, HURIER M, et al. Androzoo++: Collecting millions of android apps and their metadata for the research community[J]. arXiv preprint arXiv:1709.05281, 2017.
- [27] 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.
- [28] FELT A P, CHIN E, HANNA S, et al. Android permissions demystified[C]

- // Proceedings of the 18th ACM conference on Computer and communications security. 2011 : 627–638.
- [29] GRACE M, ZHOU Y, ZHANG Q, et al. Riskranker: scalable and accurate zero-day android malware detection[C] // Proceedings of the 10th international conference on Mobile systems, applications, and services. 2012 : 281–294.
- [30] ENCK W, GILBERT P, HAN S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems (TOCS), 2014, 32(2) : 1–29.
- [31] YAN L K, YIN H. DroidsScope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis[C] // Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12). 2012 : 569–584.
- [32] ZHOU Y, WANG Z, ZHOU W, et al. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets.[C] // NDSS : Vol 25. 2012 : 50–52.
- [33] PORTOKALIDIS G, HOMBURG P, ANAGNOSTAKIS K, et al. Paranoid Android: versatile protection for smartphones[C] // Proceedings of the 26th annual computer security applications conference. 2010 : 347–356.
- [34] CHEN S, XUE M, TANG Z, et al. Stormdroid: A streaminglized machine learning-based system for detecting android malware[C] // Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. 2016 : 377–388.
- [35] BURGUERA I, ZURUTUZA U, NADJM-TEHRANI S. Crowdroid: behavior-based malware detection system for android[C] // Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. 2011 : 15–26.
- [36] WU D-J, MAO C-H, WEI T-E, et al. Droidmat: Android malware detection through manifest and api calls tracing[C] // 2012 Seventh Asia Joint Conference on Information Security. 2012 : 62–69.
- [37] GASCON H, YAMAGUCHI F, ARP D, et al. Structural detection of android mal-

- ware using embedded call graphs[C] // Proceedings of the 2013 ACM workshop on Artificial intelligence and security. 2013 : 45 – 54.
- [38] CHAKRADEO S, REAVES B, TRAYNOR P, et al. Mast: Triage for market-scale mobile malware analysis[C] // Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks. 2013 : 13 – 24.
- [39] AAFER Y, DU W, YIN H. Droidapminer: Mining api-level features for robust malware detection in android[C] // International conference on security and privacy in communication systems. 2013 : 86 – 103.
- [40] ARP D, SPREITZENBARTH M, HUBNER M, et al. Drebin: Effective and explainable detection of android malware in your pocket.[C] // Ndss : Vol 14. 2014 : 23 – 26.
- [41] FELT A P, FINIFTER M, CHIN E, et al. A survey of mobile malware in the wild[C] // SPSM@CCS. 2011.
- [42] HU Y, WANG H, ZHOU Y, et al. Dating with Scambots: understanding the ecosystem of fraudulent dating applications[J/OL]. IEEE Transactions on Dependable and Secure Computing, 2019.  
<http://dx.doi.org/10.1109/TDSC.2019.2908939>.
- [43] CHEN S, MENG G, SU T, et al. AUSERA: Large-Scale Automated Security Risk Assessment of Global Mobile Banking Apps[J]. arXiv preprint arXiv:1805.05236, 2018.
- [44] CHEN S, SU T, FAN L, et al. Are mobile banking apps secure? what can be improved?[C] // Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018 : 797 – 802.
- [45] ZHU H, XIONG H, GE Y, et al. Discovery of ranking fraud for mobile apps[J]. IEEE Transactions on knowledge and data engineering, 2014, 27(1) : 74 – 87.
- [46] LIM E-P, NGUYEN V-A, JINDAL N, et al. Detecting product review spammers using rating behaviors[C] // Proceedings of the 19th ACM international conference on Information and knowledge management. 2010 : 939 – 948.

- [47] XIE Z, ZHU S. Grouptie: toward hidden collusion group discovery in app stores[C] // Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks. 2014: 153 – 164.
- [48] XIE Z, ZHU S, LI Q, et al. You can promote, but you can't hide: large-scale abused app detection in mobile app stores[C] // Proceedings of the 32nd Annual Conference on Computer Security Applications. 2016: 374 – 385.
- [49] CHEN H, HE D, ZHU S, et al. Toward detecting collusive ranking manipulation attackers in mobile app markets[C] // Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. 2017: 58 – 70.
- [50] HOOI B, SONG H A, BEUTEL A, et al. Fraudar: Bounding graph fraud in the face of camouflage[C] // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016: 895 – 904.
- [51] RAHMAN M, HERNANDEZ N, RECABARREN R, et al. The Art and Craft of Fraudulent App Promotion in Google Play[C] // Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 2437 – 2454.
- [52] KITCHENHAM B A, PFLEGER S L, PICKARD L M, et al. Preliminary guidelines for empirical research in software engineering[J]. IEEE Transactions on software engineering, 2002, 28(8): 721 – 734.
- [53] Cryptographic hash function[EB/OL]. 2020 [June 02, 2020].  
[https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function).
- [54] FEISTEL H. Cryptography and computer privacy[J]. Scientific american, 1973, 228(5): 15 – 23.
- [55] Apktool[EB/OL]. [June 02, 2020].  
<https://github.com/iBotPeaches/Apktool>.
- [56] dex2jar[EB/OL]. [June 02, 2020].  
<https://github.com/pxb1988/dex2jar>.

- [57] jd-gui[EB/OL]. [June 02, 2020].  
<https://github.com/java-decompiler/jd-gui>.
- [58] Android fundamentals[EB/OL]. [June 02, 2020].  
<https://developer.android.com/guide/components/fundamentals>.
- [59] AU K W Y, ZHOU Y F, HUANG Z, et al. Pscout: analyzing the android permission specification[C] // Proceedings of the 2012 ACM conference on Computer and communications security. 2012 : 217–228.
- [60] BIEMAN J M, ZHAO J X. Reuse through inheritance: A quantitative study of C++ software[J]. ACM SIGSOFT Software Engineering Notes, 1995, 20(SI) : 47–52.
- [61] HARRISON R, COUNSELL S, NITHI R. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems[J]. Journal of Systems and Software, 2000, 52(2-3) : 173 – 179.
- [62] WANG H, LI H, LI L, et al. Why are android apps removed from google play? a large-scale empirical study[C] // 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR). 2018 : 231 – 242.
- [63] DYBÅ T, DINGSØYR T. Empirical studies of agile software development: A systematic review[J]. Information and software technology, 2008, 50(9-10) : 833 – 859.
- [64] MANOTAS I, BIRD C, ZHANG R, et al. An empirical study of practitioners' perspectives on green software engineering[C] // 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). 2016 : 237 – 248.
- [65] MCINTOSH S, KAMEI Y, ADAMS B, et al. An empirical study of the impact of modern code review practices on software quality[J]. Empirical Software Engineering, 2016, 21(5) : 2146 – 2189.
- [66] MCILROY S, ALI N, HASSAN A E. Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store[J]. Empirical Software Engineering, 2016, 21(3) : 1346 – 1370.
- [67] 吴迪. 基于开源软件的 C++ 关键语言特性实证研究 [D]. 南京大学 2016.
- [68] 杨姗媛. 信息安全风险分析方法与风险感知实证研究 [D]. 中央财经大学 2015.

- [69] SEAMAN C B. Qualitative methods in empirical studies of software engineering[J]. IEEE Transactions on software engineering, 1999, 25(4) : 557 – 572.
- [70] EASTERBROOK S, SINGER J, STOREY M-A, et al. Selecting empirical methods for software engineering research[G] // Guide to advanced empirical software engineering. [S.l.] : Springer, 2008 : 285 – 311.
- [71] LEVENSHTEIN V I. Binary codes capable of correcting deletions, insertions, and reversals[C] // Soviet physics doklady : Vol 10. 1966 : 707 – 710.
- [72] Violin plot[EB/OL]. 2020 [March 27, 2020].  
[https://en.wikipedia.org/wiki/Violin\\_plot](https://en.wikipedia.org/wiki/Violin_plot).
- [73] GOOGLE. Android Security Bulletin—December 2017[EB/OL]. 2017 [September 26, 2018].  
<https://source.android.com/security/bulletin/2017-12-01>.
- [74] Baidu App Store[EB/OL]. [September 26, 2018].  
<https://shouji.baidu.com/>.
- [75] Hiapk[EB/OL]. [September 26, 2018].  
<http://apk.hiapk.com/>.
- [76] Myapp[EB/OL]. [September 26, 2018].  
<http://sj.qq.com/myapp/>.
- [77] Analysys Figure[EB/OL]. [September 26, 2018].  
<https://qianfan.analysys.cn/refine/view/rankApp/rankApp.html>.
- [78] LIFTOFF. Mobile Gaming Apps Report[EB/OL]. 2018 [September 26, 2018].  
[https://cdn2.hubspot.net/hubfs/434414/Reports/2018%20Gaming%20Apps/Liftoff\\_2018\\_Mobile\\_Gaming\\_Apps\\_Report\\_Aug.pdf](https://cdn2.hubspot.net/hubfs/434414/Reports/2018%20Gaming%20Apps/Liftoff_2018_Mobile_Gaming_Apps_Report_Aug.pdf).
- [79] VirusTotal[EB/OL]. [September 26, 2018].  
<https://www.virustotal.com/>.
- [80] WIKIPEDIA. Ransomware[EB/OL]. 2018 [September 26, 2018].  
<https://en.wikipedia.org/wiki/Ransomware>.
- [81] JONAS B. Global Malware Campaign WannaCry is Affecting Bitcoin's

- Price[EB/OL]. 2017 [September 26, 2018].  
<https://hacked.com/global-malware-campaign-wannacry-affecting-bitcoins-price/>.
- [82] COMODO. Comodo Cybersecurity Q1 2018 Global Malware Report: cybercriminals follow the money, cryptominers leap ahead of ransomware[EB/OL]. 2018 [September 26, 2018].  
<https://blog.comodo.com/comodo-news/comodo-cybersecurity-q1-2018-global-malware-report/>.
- [83] XIE Z, ZHU S. AppWatcher: Unveiling the underground market of trading mobile app reviews[C] // Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks. 2015 : 1 – 11.
- [84] ABBASI M-A, LIU H. Measuring user credibility in social media[C] // International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction. 2013 : 441 – 448.

## 攻读学位期间发表的学术论文

1. \*\*\*\*\* \*\*\*\*, \*\*\* \*\*\*\*, \*\*\*\*\* \*\*, \*\*\*\* \* \*, \*\*\*\* \*\*, \*\*\*\*\* \*\*\*\*, and  
\*\*\*\*\* \*\*, “A large-scale empirical study on industrial fake apps,” in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, IEEE Press, 2019: 183-192. (第一作者, 发表于 CCF A 类推荐学术会议, 软件工程方向顶级会议 ICSE2019)

## 致 谢

在师大的七年时光转瞬而逝，不知不觉，研究生历程已快要告一段落，我也将迎来下一个人生阶段。

借此机会，想对在师大遇上的各位老师和同学表示最真诚的感谢，尤其是我的导师，在我研究生的各个阶段都给予我鼓励和指导。同样想感谢的还有家里人和身边的朋友，是他们的支持帮助我走过了一路上的各种波折。

二〇二〇年三月