

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

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《面向工业界仿冒应用的大规模实证研究》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名:_____

日期: 年 月 日

华东师范大学学位论文著作权使用声明

《面向工业界仿冒应用的大规模实证研究》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，著作权归本人所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和学校指定的相关机构送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

- () 1. 经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文 *，于年 月 日解密，解密后适用上述授权。
() 2. 不保密，适用上述授权。

导师签名:_____

本人签名:_____

年 月 日

* “涉密”学位论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

*** 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

作为世界上最受欢迎的移动操作系统，Android 拥有着庞大而完整的应用生态环境，其中不仅包含着带给人便利的各色良性应用，也包含了不怀好意的恶意应用和意图不明的仿冒应用。得益于对恶意应用的充分理解，业界实现了对恶意应用的监控；但相对地，人们对仿冒应用所知无几，业界对其的认识匮乏则很可能会成为隐患。在尚未有前人进行系统研究的情况下，对仿冒应用进行调研十分有必要。

为了获得关于仿冒应用的第一手资料，本文直接从业界收集数据入手分析，完成本次大规模实证研究。针对现有爬虫框架不能定向爬取的问题，笔者设计实现了仿冒应用收集框架 FakeRevealer，进行了仿冒应用的大规模数据收集，将仿冒应用数据整理成集。其次，基于收集到的数据，本文分别从仿冒应用的基本特征、影响仿冒应用数量的因素以及仿冒应用的发展轨迹三个不同视角，结合实例分析，进行了首次基于 Android 系统仿冒应用的全方位特征解读。进一步地，为验证仿冒应用和移动黑灰产的另一个环节—排名欺诈之间是否存在联系，本文收集了仿冒应用在应用市场上的评级和评论进行了排名欺诈排查。针对现有排名欺诈检测方法的不足，本文先后提出两种方法创新性地对所得数据进行检测。

研究结果方面，在数据收集上，本文使用 FakeRevealer 共收集到了超过 15 万个数据条目，其中每个数据条目代表一个应用样本。利用这些样本对仿冒应用进行特征解读，本文得到了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，相关的案例分析更暴露出了仿冒作者针对不同应用进行仿冒的形式和应用市场之间的监管缺陷。这些信息都有助于对仿冒应用进行理解。评论分析方面，结合人工复核，本文确认了提出的两种检测方法都具有良好的效果，而仿冒应用中确实存在排名欺诈行为。

综上，本文分别从仿冒应用特征解读和仿冒应用评论分析两个方面完成了具

有创新性的研究。作者希望能借助本文提供的数据和分析结果，提供一个面向仿冒应用及其生态的清晰视角。

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. Comparatively, however, we barely know anything about fake apps, which may actually become a snake in the grass. Without any previous study, to conduct research on fake apps is necessary.

In order to obtain the first-hand data about fake apps, we directly dive into the industry to collect data and finish this large-scale empirical study. Considering the problem that existing crawler frameworks are not able to achieve targeted collection, the author 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. After that, based on the collected data, 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 interpretation to provide more details. 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 the result of data collection, this study obtains more than 150,000 data entries leveraging FakeRevealer, where one entry represents an app sample. Interpreting the characteristics of fake apps with these samples, this paper learns knowledge like fake apps' naming tendency and the strategies the fake app developers deploy to evade app

markets' regulatory mechanisms. What's more, the 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. On the user feedback analysis side, manual inspection infers that our two innovative detecting approaches are both effective, meanwhile, fake app developers do utilize ranking fraud to boost their rank.

This paper completes an innovative study from two aspects, namely fake apps characteristics Interpretation and fake apps user feedback 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

摘要	i
ABSTRACT	iii
目录	v
第一章 绪论	1
1.1 研究背景	1
1.2 研究现状	3
1.2.1 针对恶意应用生态系统的研究	3
1.2.2 重打包应用检测研究	3
1.2.3 灰色应用的实证研究	4
1.2.4 应用评论相关研究	4
1.3 研究意义与研究难点分析	5
1.4 工作概览	6
1.5 本文组织结构	7
第二章 移动端黑灰色产业生态分析	9
2.1 Android 应用程序的构建与签名	9
2.1.1 Android 应用程序构建	9
2.1.2 Android 应用签名机制	11
2.2 Android 应用市场与移动黑灰产	12
2.3 移动黑灰产简介	13
2.3.1 恶意应用	13
2.3.2 排名欺诈	15

2.4	本章小结	16
第三章	面向仿冒应用的收集框架 FakeRevealer	17
3.1	设计缘由	17
3.2	应用收集流程简介	17
3.3	FakeRevealer 的设计与实现	18
3.3.1	应用收集器	18
3.3.2	迭代搜索器	20
3.3.3	应用过滤器	22
3.4	仿冒应用数据概览	22
3.5	本章小结	23
第四章	结合实例分析的仿冒应用特征解读	24
4.1	研究概况	24
4.2	仿冒应用的基本特征	24
4.2.1	安全证书与仿冒应用的对应关系	24
4.2.2	仿冒样本与原版应用的相似度	25
4.2.3	案例 1. 持有官方安全证书的可疑样本	29
4.3	影响仿冒应用数量的因素	31
4.3.1	仿冒应用的来源	32
4.3.2	其他因素对仿冒样本数量的影响	33
4.3.3	案例 2. 游戏类别下的仿冒应用	37
4.4	仿冒应用的发展轨迹	39
4.4.1	仿冒应用的研发延迟	39
4.4.2	仿冒应用安全证书的存活时长	41
4.4.3	仿冒应用的行业变迁	43
4.4.4	案例 3. 与大量仿冒应用相关联的仿冒应用安全证书	43

4.5	本章小结	45
第五章	面向仿冒应用的排名欺诈验证方法	46
5.1	研究概况	46
5.2	仿冒评论数据收集	46
5.2.1	评论数据概览	47
5.2.2	基础分析	47
5.3	仿冒应用与排名欺诈关联验证	49
5.3.1	排名欺诈检测初探	49
5.3.2	基于评论用户可信度的排名欺诈排查	50
5.3.3	基于评论内容相似度的排名欺诈排查	52
5.3.4	人工复核	55
5.4	本章小结	55
第六章	总结与展望	58
6.1	总结	58
6.2	展望	59
参考文献	61	
攻读学位期间发表的学术论文	67	
致谢	68	

插 图

图 1.1 Google Play 应用商店架上应用总数变化趋势	1
图 1.2 本文工作概览	6
图 2.1 Android App 构建流程	10
图 2.2 更新攻击样本示例	14
图 3.1 Janus 平台上的数据规模时序图	18
图 3.2 FakeRevealer 整体结构	19
图 4.1 对 App 各项属性的统计结果	27
图 4.2 各大应用市场应用详情页（从桌面端浏览）	29
图 4.3 从不同应用市场中收集到的应用数量以及各市场仿冒率	32
图 4.4 游戏类 App 及其仿冒样本	37
图 4.5 仿冒延迟总体分布	40
图 4.6 仿冒应用安全证书存活时间分布	42
图 4.7 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）	42
图 5.1 各仿冒应用在 360 手机助手商店中的评论数量与评分分布	47
图 5.2 所有 856 个应用在商店中的评论数量与评分分布	48
图 5.3 两种数据组的应用评论重合率结果	54
图 5.4 应用评论取样	56

表 格

表 4.1 安全证书/仿冒应用数量对应表	25
表 4.2 持有官方安全证书的可疑样本	30
表 4.3 目标 App 与其相关统计	35
表 4.4 由“61ed377e85d386a8dfee6b864bd85b0bfaa5af81”签署的部分样本 .	44
表 5.1 各应用用户可信度权重及对应排名（一）	52
表 5.2 各应用用户可信度权重及对应排名（二）	52
表 5.3 评论重合率结果	54

算 法

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。根据数据分析机构 StatCounter 的资料显示，从发布之日起，Android 的市场占有率就在逐年稳步增长。截至 2020 年，Android 系统已经占据了 74.3% 的全球移动端市场份额^[1]。与此同时，Android 应用的数量也伴随着 Android 市场的蓬勃发展节节攀高。仅看 Android 官方的应用商店 Google Play，其在 2017 年一年中就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^[2]。

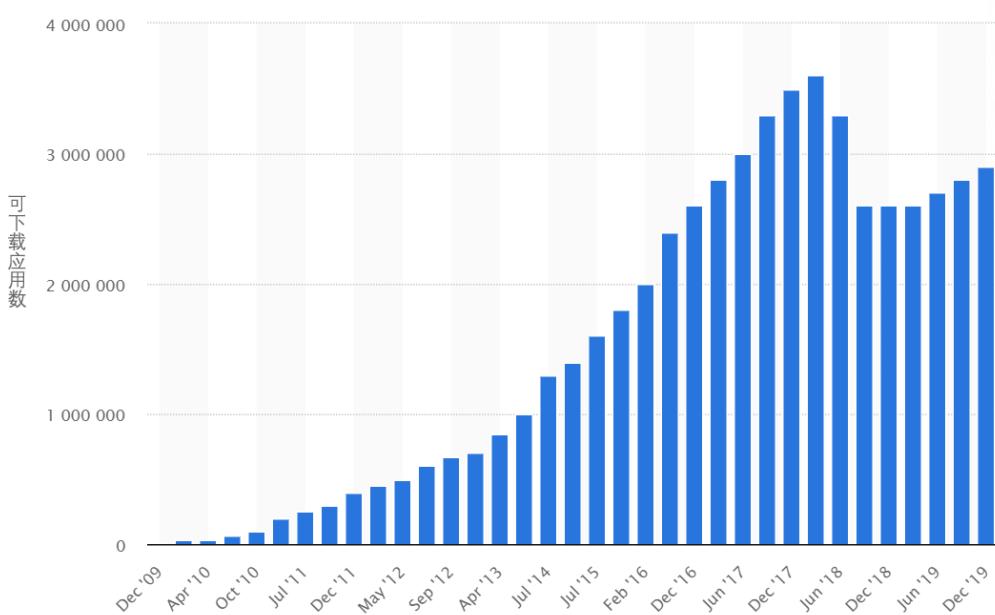


图 1.1: Google Play 应用商店架上应用总数变化趋势

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。仿冒应用构建了移动黑灰产业链中十分重要的一环。本文提及的仿冒应用指代模仿市面上热门的

应用、甚至外观与热门应用相差无几的移动应用，其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。根据的前期观察，笔者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^[3,4]，已不单单是与原应用“相似”了，这类应用采用和原应用一模一样的外观，乃至连版本号都相同，其中的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。可以假设一个十分普遍的场景：当用户尝试通过应用市场搜索安装一个应用，应用市场往往会返回多个无论是名字或是图标都十分相像的结果，在这种情况下，用户十分有可能安装一个仿冒应用。就算用户最后通过某些途径安装上了原版应用，也浪费了时间和人力成本，遑论某些仿冒应用中潜藏着恶意行为，一旦被触发就会对用户造成更大的损害。于是，用户搜索与下载应用时的安全和体验就被仿冒应用严重影响了。

更糟糕的是，随着开发移动应用的门槛逐渐下降，开发一个仿冒应用的成本已经远低于开发一个桌面级应用所需的成本，这为地下产业涌入移动端发展提供了绝佳的温床^[5]。此外，移动应用功能在实现上的灵活性^[6]也增加了仿冒应用的复杂度，让分析移动应用变得更加困难。

尽管如今仿冒应用随处可见，业界和学术界对仿冒应用和他们的生态却依然知之甚少——他们有何共同特征、数量多少、迭代速度如何、以及他们如何规避应用市场检测等问题依然有待解答。当前关于移动应用的学术研究大多关注于恶意应用的检测技术^[7-10]。据笔者所知，目前还未有任何关于仿冒应用或其生态系统的研究。类似地，工业界中也鲜见关于仿冒应用的课题。多数分析与威胁报告都聚焦于恶意应用上，忽略了仿冒应用^[11]。而在另一方面，由于底层、实现等各个层面上的差异，从桌面平台上获取的关于仿冒软件的知识也不能很好地用于了解移动端仿冒应用^[12]。

1.2 研究现状

1.2.1 针对恶意应用生态系统的研究

在 Felt 主导的一次研究^[13] 中，研究人员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。该篇文献也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。另外，Zhou 和 Jiang^[14] 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，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）；在另一篇著作^[15] 中，他们提出了名为 DroidRanger 的系统，并成功地从 5 个应用市场的 204,040 个应用中找出了 211 个恶意应用。

这类研究帮助了业界人员拓宽视野，使得业界人员对恶意应用的行为更加了解。然而正如上文提及，仿冒应用与恶意应用分属同一产业中的不同链条，针对仿冒应用的专门研究依然是有必要的。

1.2.2 重打包应用检测研究

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，重打包应用也属于移动黑灰产业范畴。针对重打包检测的前人研究大致可划分为五个类别：

第一类是基于应用指令序列的。这种方法使用模糊哈希的方法提取出应用的摘要信息，然后通过比对两两应用之间的摘要信息来获得应用之间的相似度^[16,17]。第二类凭借语义信息比对应用。如 CLANdroid^[18] 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），检测相似的应用。第三类利用了第三方库进行检测。CodeMatch^[19] 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。Wukong^[20] 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。ViewDroid^[21] 通过重建和比对应用的视图来筛出重打包应用，这种技术属于第四类—信息可视化。第五类依赖图论衡量应用相似性。DNADroid^[22] 基于应用的程序依赖图 (Program Dependency Graph, PDG) 来比对应用相似性，而 AnDarwin^[23] 则用从每个方法中提

取的 PDG 构建出语义向量，再计算向量间相似度以检测重打包应用。Centroid^[24]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)，然后再将三维控制流图聚合，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。

检测方法林林总总，在此不一一列举，从检测的准确性和可伸缩性上看，每种方法均有优缺点，其并不在本文的讨论范围之内。然而，他们全都有一个共同之处：在验证阶段，无一例外，都是基于证书系统进行的。一旦非法开发者利用具有漏洞的证书签名机制，污染了实验中的部分应用数据，即使是最先进的重打包检测机制也无用武之地。

1.2.3 灰色应用的实证研究

Andow 等人曾发表过一篇针对灰色应用的研究文献^[3]。其中，他们从 Google Play 应用商店中采集了应用样本，并将样本分类，定义出了 9 种不同的灰色应用。灰色应用即那些并非具有明显的恶意行为，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。本文中对高仿应用的定义参考了这篇文献中的内容。

1.2.4 应用评论相关研究

近年来，除了对 Android App 本身的分析研究之外，有不少学者也把研究焦点投放在了应用市场的评论上。

Bin Fu 等人在 2013 年发布了 WisCom^[25]—对于某个应用，将其历史评级和评论记录输入 WisCom，系统在反馈应用评级变化的同时，也能告诉使用者为什么会影响到恶评；而 Deguang Kong^[26] 等人在 2015 年发布的 AUTOREB 则结合了自然语言处理、机器学习技术和声誉系统，先提取用户在某款 App 的评论中提及到的与安全相关的内容，然后根据每个用户的可信度对该款 App 是否可能有危险行为进行分数计算，最后用机器学习技术对 App 进行分类，预测该款 App 是否会有危险行为。

另一方面，大量的前人工作^[27–33] 揭示，很多应用市场都饱受虚假评论的困扰。Rahman 等人在 2019 年发布了一个关于虚假评论行为的实证研究^[34]，在公开确认

了这个产业链存在的同时，也揭露了他们的行为模式、生存情况甚至是收入水平等方面的信息。

1.3 研究意义与研究难点分析

前文的研究现状，不仅反映出移动黑灰产业是目前的研究热点之一，更反映出移动黑灰产从业人员无孔不入的特点。为了保障正当开发者的利益与消费者的权益，学术界和工业界都需要对移动黑灰产有更全面、更深入的理解，从而更好地预防未知的威胁。然而，现有研究提供的知识仍有空缺部分，仿冒应用部分正是缺口之一。仿冒应用有什么样的形态？仿冒应用的发展有什么样的趋势？仿冒应用的市场反馈如何？这一环节会与其他黑灰产业环节有关联吗？

为了回答这些疑问，笔者与肆众信息的移动安全威胁数据平台 Janus^[35] 合作，搜集并分析了大量应用样本，从仿冒应用特征解读和仿冒应用评论分析两方面进行了大规模实证研究，对仿冒应用作出了较为全面的剖析，填补了本领域的研究空白。仿冒应用特征解读利用数据挖掘分析技巧，利用本研究收集到的仿冒应用数据对仿冒应用进行画像；仿冒应用评论分析则通过收集仿冒应用在市场上获得的反馈，验证仿冒应用和排名欺诈行为的关系，进一步地提供了关于仿冒应用生态的信息。

在数据挖掘分析中，研究者通常都会遇到几点挑战：如何确定研究主体、如何收集数据、如何对数据进行有效处理；而在排名欺诈相关研究中，如何从评论数据中有效挖掘排名欺诈行为是研究者经常要思考的问题。因此，本实证研究中的难点主要包括如下几点：

1) 如何确定仿冒应用 仿冒应用和正版应用是相对的概念。笔者选择了市面上最热门的 50 个应用，再搜集其对应的仿冒应用样本。从应用中筛选出与热门应用外观相似或是相同的样本后，本文使用 Android 本身自带的证书机制，将获得样本的证书信息与原版应用的证书信息进行比对，从而鉴别出仿冒的样本。

2) 如何获得针对仿冒应用的大量数据 数据搜集是科研工作中公认的难点。本文想要提供一次全面的研究结果，除了搜集的目标应用需要有多样性之外，也必须获得不同应用市场上的数据，增加研究的代表性。前文提及肆众信息的移动安

全威胁数据平台 Janus 是一个数据整合平台。该平台每天从各大 Android 应用市场爬取应用样本入库，免去了要针对各个市场重新定制爬虫代码的麻烦。通过设计和实现仿冒应用收集框架 FakeRevealer，笔者顺利从 Janus 搜寻到了超过 15 万条数据条目作为原始数据，其中每条数据条目代表 Janus 从应用市场上获得的一个应用样本。

3) 如何对大量的数据进行有效处理 数据规模和处理效率一直是一对矛盾。由于一条数据条目代表一个应用样本，要对超过 15 万个应用样本进行详尽分析，明显太耗费时间成本与计算成本；然而，如果只对样本进行简单处理，获得的分析结果就不够全面和深入。在尽量确保分析全面性的前提下，对于每个样本，本研究只抽取 8 个关键信息项进行分析，以节省时间与计算成本。

4) 如何挖掘评论中的排名欺诈行为 现有的排名欺诈挖掘工作均具有其各自的局限性，或是对数据有连续收集要求，或是要求检测者有已知的排名欺诈群体，并不能直接应用到本研究收集的数据中。为此，本研究先后设计了基于用户行为可信度和基于评论内容重复度的两个方法。前者规避了现有方法的局限性，后者进一步解决了大数据量带来的大运算量问题。

1.4 工作概览

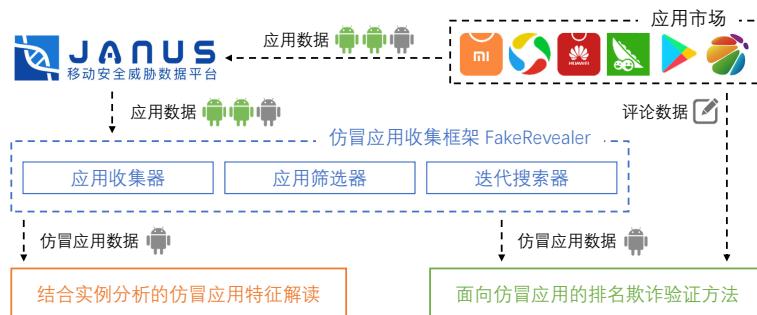


图 1.2: 本文工作概览

如图 1.2 所示，本工作通过三个主要部分完成实证研究：

1) 面向仿冒应用的收集框架 *FakeRevealer* 针对现有爬虫框架不能定向爬取应用的问题，本研究设计并实现了仿冒应用收集框架 *FakeRevealer*，以进行仿冒应用的数据收集。数据收集主要分为两个部分：正版应用信息的收集和仿冒应用

的收集。在正版应用信息收集的部分，本文选择了 50 个最热门的 App 作为目标 App，然后手动收集了跟这些 App 有关的信息；仿冒应用信息收集方面，本研究利用 FakeRevealer，通过 Janus 平台收集了从各个应用商店获得的大量仿冒应用样本。

2) 结合实例分析的仿冒应用特征解读 针对移动黑灰产研究中关于仿冒应用的研究空白，本研究收集仿冒应用数据，结合案例分析，进行了首次基于 Android 系统仿冒应用的特征解读。特征解读从三个视角完成，这些视角分别是仿冒的基本应用特征、影响仿冒应用数量的因素和仿冒应用的发展轨迹，由浅入深揭示仿冒应用的生态。对应的三个案例分析除了为特征解读提供实例支持之外，还揭示了更多仿冒应用开发者的行为特征。

3) 面向仿冒应用的排名欺诈验证方法 在这个部分，本文从第三方应用市场中随机选取一部分应用，爬取了用户对它们的所有历史评价，以检测仿冒应用与排名欺诈行为的关联。针对前人检测研究需要先验知识或特殊数据的问题，本研究从社交媒体研究引入了用户行为可信度进行排查，避开了现有方法的局限性。进一步地，本研究从评论内容重复率方面提出了另一创新性排查方法，解决了数据量增大带来的大运算量问题。人工复查结果显示，两种排查方法均取得了优于现有方法的结果。

1.5 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、研究意义和工作概览。

第二章 介绍了 Android 应用的构建流程、Android 安全证书机制、应用市场与黑灰产业的关系和一些移动黑灰产知识。

第三章 阐述了仿冒应用收集框架 FakeRevealer 的设计缘由，说明了仿冒应用的数据收集方式并详细介绍了其中三个组件（应用收集器、迭代搜索器和应用过滤器）的设计与实现，最后提供仿冒应用数据概览。

第四章 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。这些视角包括仿冒应用特征、影响仿冒应用数量的因素和仿冒应用的发展轨迹，每个

视角都被进一步分解成了多个不同的研究问题。在完成每个视角的解读后，笔者均从数据中挑选一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。

第五章 提供了面向仿冒应用的排名欺诈检测方法。针对现有检测方法的不足，笔者先后提出了两个具有创新性的方法对排名欺诈行为进行排查。结果显示，仿冒应用与排名欺诈行为作为移动黑灰产的两个环节确实存在关联。

第六章 对本文工作进行总结，并对下一步工作进行展望。

第二章 移动端黑灰色产业生态分析

随着人们对电子产品的焦点从桌面端慢慢转移到移动端，互联网黑灰产业也有向移动端趋向的迹象。本章将按自底向上的顺序，分别介绍 Android 应用程序¹、应用市场和移动黑灰产相关的背景知识，以期让读者对移动黑灰产的整体生态有大致的了解。

2.1 Android 应用程序的构建与签名

Android 应用是移动黑灰产的终端环节之一，了解 Android 应用的构建和签名过程能帮助业界和学界了解移动黑灰产从业人员可以在哪一步植入牟利手段。

2.1.1 Android 应用程序构建

与大部分软件一样，开发者在发布自己的 App 之前，都需要先把代码编译打包成 Android 操作系统使用的一种应用程序包格式文件，即 APK (Android application package)。每个 APK 文件都会包含该款 App 的一系列基本信息，包括 App 的应用名、包名 (Package name)、安全证书等。其中，包名是 Android 系统识别 App 的依据，每款 App 在不同的版本可以有不同的应用名，但其包名必须是一致的。图 2.1 展现了 APK 文件的构建流程。一般来说，一个 Android App 的构建流程会分为以下四步，整个构建流程由 Android SDK 中的 Android 插件和 Gradle 构建工具管理。

首先，开发者需要编写 App 对应的源代码，然后连同一些源代码中使用到的依赖项一起输入到编译器中生成 DEX 文件。此外，编译器还会将其他未被编译的资源文件转换为编译后的资源。黑灰色产业从业者可以在这步就将恶意代码植入到应用中。

然后，SDK 中的 APK 打包器会将 DEX 文件和已经编译好的资源文件一起打包。APK 文件的本质是压缩文件，所以 APK 打包器的任务是将所有相关文件都压

¹应用、App 在本文中均用以指代 Android 应用程序，不再区分

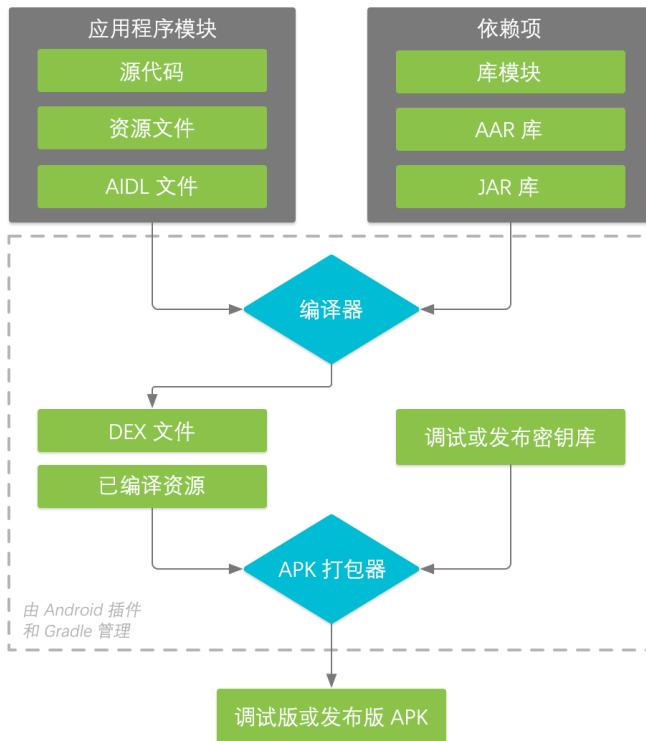


图 2.1: Android App 构建流程

缩进一个 APK 文件里面。不过，在这个步骤，APK 打包器还未将所有文件压缩。因为在压缩之前，还需要进行下一步的签名。

在第三步，APK 打包器会使用密钥库文件对上一步中提及的资源文件和代码文件进行数字签名。这个步骤是用作校验 APK 文件是否被篡改、保证 APK 文件完整性的一个重要步骤，但也是黑灰色产业从业者可以利用的一个环节。下一小节会有相关机制的更多介绍。

最后，APK 打包器会使用 zipalign 工具对应用进行优化，以减少 App 在设备上运行时所占用的内存。这步结束之后，整个构建流程也随之结束。开发者会获得一个编译好、签名完毕并且经过优化的 APK 压缩文件，然后就可以将这个 APK 文件安装到 Android 设备上运行使用。

目前，随着技术发展，互联网上还出现了各式各样的 App 生成器^[36,37]，用户无需了解复杂编程知识，只需简单操作即可实现 App 的开发。一方面，这种简易的开发步骤有助于移动黑灰产从业人员快速生产恶意应用；另一方面，App 开发框架的提供者本身也可能对生产出的 App 嵌入恶意代码。业界相关报告^[38]表明，移

动黑灰产从业人员对 App 生成器的滥用已经带来了严重的安全隐患。

2.1.2 Android 应用签名机制

开发者在使用 Android SDK 构建 App 时，其中十分重要的一步是对 App 进行数字签名。实际上，Android 的数字签名和安全证书机制基于 RSA 公共密钥系统，是 Android 安全机制中不可或缺的一个部分。

Android App 的签名机制是用作校验 APK 文件是否被篡改、保证 APK 文件完整性的一个重要机制，所有的应用都必须要在经过签名之后才能安装进 Android 系统中。在签名时，SDK 会使用一种密钥库文件，如果开发者还没有这个文件的话，SDK 会自动生成一个。密钥库中包含了开发者的各种信息，包括一对公钥和私钥。私钥用于数字签名，不可向外公布；公钥则是可以向外公布的一组密钥，用于数字前面的验证。App 中的签名也是系统用来识别开发者的重要依据，因为同一个密钥库文件会产生一致的签名，系统能根据签名中的公钥验证应用识别开发者。

在一个 APK 被打包签名完毕之后，如果需要更改其中的内容，就只能在更改后将 APK 重新打包签名一次，即使是一个 bit 的修改也会破坏原有的签名，导致安装时验证签名失败。在安装 App 的过程中，验证签名失败会使得系统终止 App 的安装。这也是系统可以用数字签名识别开发者的原因：签名一致的 App，最后一定都是由同一个开发者打包的。所以，具有同样签名的 App 可以在同一个 Android 设备上共享数据。不过这超出了本文讨论的内容，故不再赘述。

目前，签名的模式共有三代，其区别主要在于构建流程第三、第四步之间的一些操作上。简单地说，越新的签名模式能更好地保障 APK 文件的完整性。实际上，第一代签名模式 V1 具有较为致命的缺陷，所以 Google 官方也在呼吁开发者在编译时采用最新的签名模式。

要注意的是，签名机制只能保证 APK 文件在被篡改之后不能凭借原有的签名被安装进 Android 系统，但恶意开发者依然可以在篡改 APK 之后，用自己的密钥库对 APK 重新签名，构建出可安装的 App。这种 App 是盗版 App 的一种，被称为重打包 App。

另外，虽然一个安全证书只能指向一名开发者，但一个开发者可以同时拥有多

个安全证书。开发者和安全证书之间具有一对多的映射关系。

2.2 Android 应用市场与移动黑灰产

由于每个人都可以开发、构建自己的 Android App，从网上发布的 App 数不胜数。这为 Android 应用生态带来开放性的同时，也会引入包括安全隐患等一系列问题，而 Google 提供的 Google Play 应用商店无疑为用户和开发者都提供了一个优良的解决方案。官方对上架前的应用审核，为用户安全提供了一定保障；商店中每个应用底下由用户评论组成的社区也促成了用户和开发者之间的交流，用户反馈直接推动了开发者对应用的改良。

遗憾的是，由于种种原因，Google Play 应用商店的服务并非对全球的所有地区和国家都开放。Google 从 2008 年开始退出中国大陆市场，因此 Google 的大部分服务，包括 Google Play 应用商店的下载服务在内，都不向中国大陆境内用户提供。换句话说，国内的大部分普通用户并不能享受到 Google Play 应用商店的便利。

为此，国内有多家厂商都推出了自己开发的应用市场服务，如腾讯旗下的应用宝^[39] 和百度旗下的百度应用市场^[40]，还有华为的应用市场、小米的小米应用市场^[41] 等，试图填补这一片市场空白。

不同于在 Android 系统发布早期就存在的 Google Play 应用商店，国内的第三方应用商店是后期出现的产物，一出现就面临着激烈的市场竞争。一方面，在国内各类第三方应用市场方兴未艾之时，国内的 Android 开发者社群尚未成熟，应用市场还未有大量开发者进驻；另一方面，在成立初期，为了抢占市场份额，各个应用商店都想方设法将商店内 App 的种类和数量最大化，以迎合市场用户各种各样的需求。作为结果，各类第三方应用市场都在各个渠道搜集 App，而非通过开发者上传的方式获得货架上的应用程序。由于在早期各种监管渠道、审核机制尚未完善，各个市场在搜罗各类 App 的同时，难免会将大量的盗版、恶意应用也一并收录。换句话说，早期的各类第三方市场为移动黑灰产的成长提供了良好的温床。

2.3 移动黑灰产简介

依托 Android 端应用进行牟利的移动端黑灰色产业链条多种多样，在此本节介绍与本次研究较为相关的两种：恶意应用与排名欺诈。

2.3.1 恶意应用

恶意应用是移动黑灰产中最重要的一环。作为与用户接触的终端，恶意应用也是多种形式的移动黑灰产的负载触发点。

要了解恶意应用产业，可以从三个问题下手：恶意应用是怎么安装到用户的设备里的？恶意应用都会有什么恶意行为？这些恶意行为是如何触发的？

1) 恶意应用的安装 相关研究^[14]表明，恶意软件的安装途径主要可以分为三个分类：重打包、更新攻击和路过式下载（Drive-by Download）。

重打包的概念在第 1.2.2 节已经有所介绍。为了诱使用户下载重打包后的应用，移动黑灰产从业人员往往会选择热门的应用进行重打包，再在监管力度不太严格的应用市场进行重新发布。这意味着，在本研究的主体—仿冒应用中，会有一部分重打包应用。但是，迄今并没有相关研究揭示仿冒应用中重打包应用的占比，笔者将在后文进行研究。

更新攻击是指恶意应用开发者为了躲避应用市场的监管审查，有意在应用市场上上架不包含恶意代码的应用，然后在用户安装应用之后，提示用户升级，进而绕开应用市场，将带有恶意代码的“新”版本安装到用户设备上的行为。图 2.2 给出了一个更新攻击的样本示例，图片引自 Zhou 与 Jiang 于 2012 年发布的研究^[14]。

比起上述的两种安装途径，路过式下载更加隐秘。路过式下载指在用户不知晓的情况下下载和安装恶意软件，和更新攻击相似，路过式下载行为的携带者（也就是一个表面上没有恶意代码的应用）可以被正常地上传到应用商店中，供用户下载。等用户安装了对应应用之后，就有可能因为各种事件触发路过式下载，将不想要的应用甚至其他恶意应用静默安装到设备上。这些事件可以是误触了应用中的某个弹窗广告，也可以仅仅是打开了无线网络开关，路过式下载甚至可以发生在设备每次启动完毕的时候。

2) 恶意行为分类 恶意应用包含的恶意行为同样具有多样性。以对设备侵入

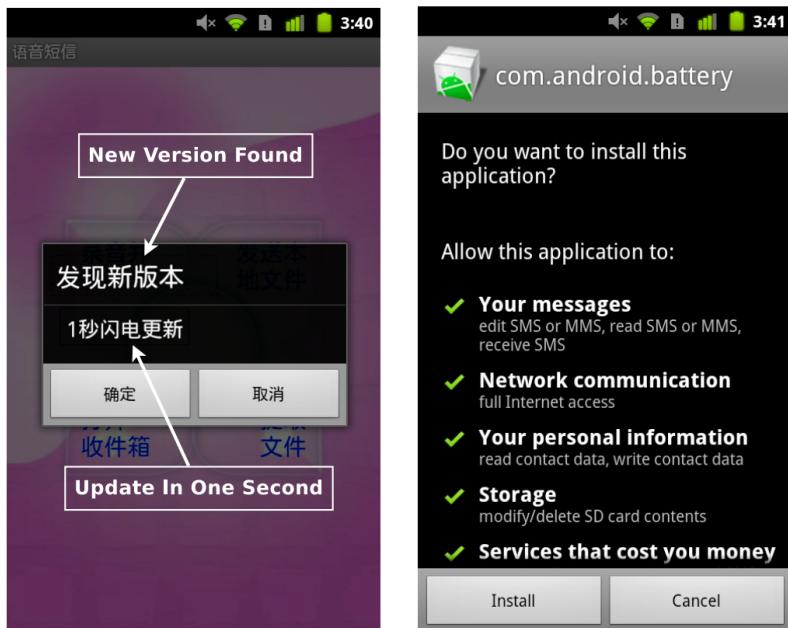


图 2.2: 更新攻击样本示例

性从高到低排序，已知恶意行为可以被分为几个分类：特权提升，远程控制，恶意扣费和信息收集。

特权提升指恶意应用利用系统级别漏洞，获取其原本不应有、甚至超越用户级别的权限。一旦有了这些权限，恶意应用就能对用户设备中的数据进行任意篡改，获取到系统级别权限的恶意应用甚至可以对用户设备进行任意操控，危害用户的数据安全。具有这类行为的代表应用有形形色色来源不明的 Root 软件。

远程控制则利用了远程服务器对用户设备进行操纵。具有这类恶意行为的应用都是木马程序，他们往往会在代码中隐含着一个 C&C 服务器地址（命令和控制服务器，Command and Control Server）。应用启动之后，就会在后台与服务器联系，接收并执行来自服务器的命令。前段时间操控用户设备的 Android 挖矿应用就可以归入具有这类行为的应用中。

恶意扣费行为通常与电信服务运营商相关的订阅服务和短信收发权限有关。在这类行为被触发时，恶意应用会在后台利用发送短信的权限向运营商发送服务订阅短信，部分恶意应用甚至会拦截运营商的订阅确认短信，对其屏蔽或者删除，使用户在不知情的状况下消耗手机资费。这类行为的代表应用有被重打包的小游戏或者盗版的视频播放器。

信息收集行为最为普遍。进行信息收集的应用会利用获得的权限搜集用户设备中的私人信息，如通讯录信息、位置信息、设备的唯一识别码（IMEI）等，再返回给恶意开发者。恶意开发者可以对这些信息进行倒卖获利，也可以利用这类信息对手机用户精确画像，对用户进行精准营销。

3) 负载触发条件 借助 Android 系统中各组件之间灵活多变的沟通手段，恶意应用负载（即恶意行为）的触发方式也有不同的种类。

以路过式下载触发方式为例，前文提及的点击应用内弹窗广告导致路过式下载可以算是主动触发方式，因为这需要用户主动点击应用中的控件才能触发；打开无线网络开关和设备启动完毕就导致的路过式下载则属于利用应用监听系统广播触发的被动触发方式，这类应用会在配置文件中注册监听器，接收来自系统的广播信息，一旦捕获到相关的广播信息，就会运行恶意代码。

恶意应用的负载触发次数越多，恶意开发者的获利可能就越大，但同时恶意行为暴露的可能性也会越大。为了躲避查杀和被用户察觉，有些恶意应用还会选择牺牲一定的触发频率，挑选十分苛刻的条件触发负载。Pandita 等人在 2013 发表的研究 WHYPER^[42] 就寻找到了一个触发条件苛刻的应用。该应用只会在半夜 12 点后，在设备收到的移动网络信号发生变化时才会执行负载。

上述背景知识表明业界对恶意应用的特征认识已经较为充分。但是，人们对仿冒应用的特征却所知无几，因此，对仿冒应用进行特征解读十分有必要。

2.3.2 排名欺诈

应用市场的评论系统营造了一个社区环境，搭建了用户和开发者之间交流的平台，其他用户也能从其他用户的评价决定自己是否也要下载某款应用。能从其他用户的反馈中获得参考固然是好事情，不少开发者也的确从用户的评论内容中得到了启发。但不幸的是，移动黑灰产从业人员从评论区中也发掘出了商机。

一些移动黑灰产从业人员以应用搜索优化/应用商店优化 (App Search Optimization/App Store Optimization，简称 ASO) 的名义，为一些不良商家提供操纵应用榜单排名的服务（即排名欺诈，Ranking Fraud）。他们会为客户的 App 提供大量虚构的五星好评，刷高这些 App 的评分与在应用商店中的排名，而不管该款应用

质量如何。这种行为在令一些本身质优的应用被埋没的同时，也让一些广受好评的应用陷入信任危机，破坏了正常的市场秩序。

排名欺诈与仿冒应用是移动黑灰产生态中的两个独立环节，为了验证两者之间是否存在关联，本文开展了评论分析研究。

2.4 本章小结

本章主要介绍了 Android 应用的构建流程、Android 安全证书机制、应用市场与黑灰产业的关系和一些移动黑灰产知识。移动黑灰产从业人员通过直接编写恶意代码或者篡改原版 APK 的方式生产恶意应用，而国内早期第三方市场的监管不完善助力了移动黑灰产业的成长。如果以恶意应用为代表的黑灰产环节进一步利用了排名欺诈行为提升其影响力，那么受其侵害的用户群体将很有可能被进一步扩大。

第三章 面向仿冒应用的收集框架 **FakeRevealer**

3.1 设计缘由

鉴于目前学术界中未有相关工作能提供仿冒应用的相关数据集，本研究需要率先尝试从工业界中系统地收集所需的仿冒应用数据。然而，传统的网络爬虫框架只能机械、不加区分地爬取数据，不能具有针对性地爬取与本研究中的目标应用相关的样本；应用市场中应用数量太多，完全收集并不可行。为了解决这个问题，本研究提出了面向仿冒应用的收集框架 **FakeRevealer**，创新地采用启发式的方法搜索与爬取与目标应用相关的样本，为后文的两项研究服务。

3.2 应用收集流程简介

仿冒应用是一个跟正版应用相对的概念，所以本文需要先定义正版应用，再根据正版应用的信息搜寻仿冒应用。

1) 正版应用收集 在定义正版应用方面，本研究参考了数据平台易观千帆的月度 App 排行榜^[43]，然后从中选出了其中的前 50 款热门 App 作为本次研究的目标 App。之后，笔者手动地从这 50 款 App 的官方网站上下载了这些应用的最新版本，作为正版应用的参考版本。

2) 仿冒应用收集 要获取足量的仿冒应用数据以组成数据集是一个十分具有挑战性的任务，难点如下：

- 要从多个不同的应用市场中爬取 App 样本，但每个应用市场都有不同的网页编码，不存在一个爬虫脚本对所有应用市场数据都通用的场景；
- 各个应用市场架上的 App 数量浩如繁星，需要有效地找到和目标 App 有关的所有样本，不重不漏；
- 对于大量数据，需要一个轻量级的解决方案快速判断获得的 App 样本究竟为

正版应用又或者是仿冒应用。

为了应对第一个挑战，本研究与工业界合作，利用舜众信息公司的 Janus 平台对各大应用商店进行样本爬取，从而获得大量应用样本。事实上，如图 3.1 显示，Janus 平台自 2017 年起就开始对各大应用商店的 App 进行样本收集，至今已收集到上千万个 App 样本。除样本搜集外，Janus 也提供按规则搜索功能，用户可以创建自己的规则过滤平台中的应用数据，以获取自己需要的 App 样本。

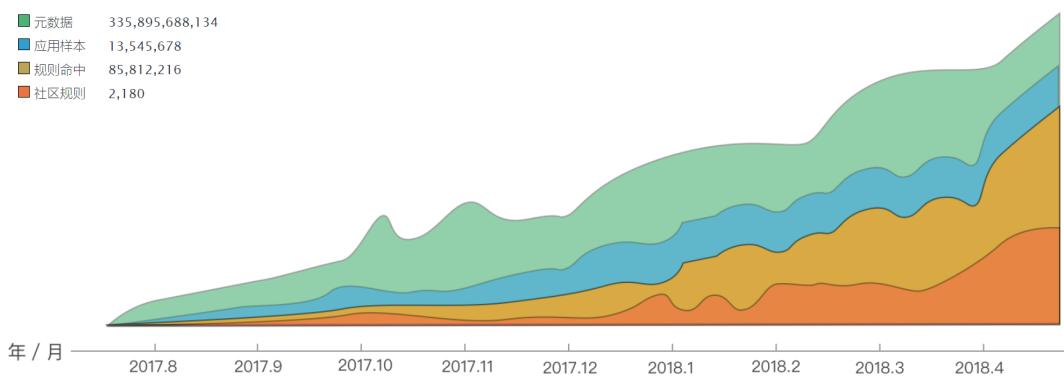


图 3.1: Janus 平台上的数据规模时序图

为了应对第二个和第三个挑战，本研究搭建了面向仿冒应用的收集框架 FakeRevealer。

3.3 FakeRevealer 的设计与实现

本文使用 Python 3 开发了仿冒应用收集框架 FakeRevealer，该框架由应用收集器、迭代搜索器和应用过滤器三个组件组成，图 3.2 展示了 FakeRevealer 的整体流程图。输入初始正版应用的信息，FakeRevealer 在经过迭代搜索、样本下载、应用过滤三个步骤之后，以 CSV 文件和 JSON 文件的形式输出仿冒应用各数据项和拓展后的正版应用信息。

3.3.1 应用收集器

应用收集器是与应用来源直接交互的部分，分为两个不同的子模块，分别是网络爬虫模块和 APK 包预处理模块。在接收到应用收集请求之后，应用收集器会先

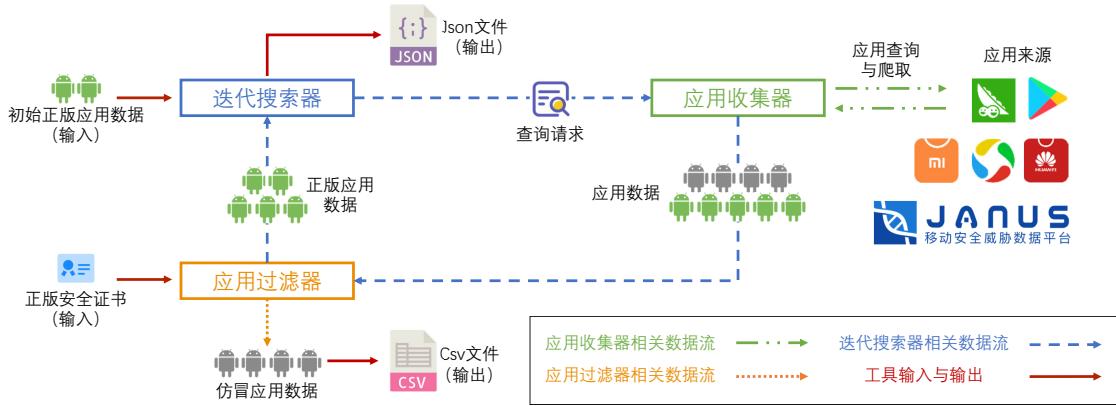


图 3.2: FakeRevealer 整体结构

根据收集请求，利用网络爬虫模块下载应用，然后再用 APK 包预处理模块对下载完毕的 APK 包提取信息，方便后续操作。

1) 网络爬虫模块 本模块负责接收应用收集器的输入，根据输入中的需求从应用来源中查询并下载对应的 App。由于 Janus 平台上已经有提前收集好的源于各个应用市场的 App 样本，笔者直接从 Janus 上爬取应用。

不同应用商店提供应用查询、下载的 API 不同，不存在可以对应所有应用市场的爬虫脚本。但只要研究者能分析出某应用商店查询、下载 API 的名称和用法，结合应用商店账户的 Cookies，就能从该商店中下载应用。考虑到这个方面，笔者开发了插件化的爬虫模块。针对不同的应用商店，使用者可以在使用网络包分析工具（如 Burpsuite^[44]）解析到 API 与 Cookies 信息之后，将对应 API 和 Cookies 写入配置插件，再对网络爬虫模块配置对当前商店的插件，开始爬取应用。

在具体实现方面，笔者利用 Python 自带的 urllib 库实现对网络资源的访问；由于下载是可以并发进行的事务，为了能提高运行效率，笔者利用了 threading 库对网络爬虫模块提供了多线程特性。

2) APK 包预处理模块 这个模块负责从下载完毕的 APK 包中提取指定的数据项，存入键值对中，最后将所有应用的数据键值对以列表形式返回，作为应用收集器的输出。APK 文件中虽然包含着应用的所有信息，但本研究中的应用筛选并不需要用到整个 APK 文件，所以本模块会把筛选需要用到的数据先提取出来，后续处理时直接调用与该 APK 包有关的数据即可。同时，由于不同 APK 包的大小不

一，所需下载时长也各异，本框架在较大的应用还在下载的同时，对已经下载完毕的 APK 提取信息。比起让应用收集器直接返回下载完毕的 APK 包、待后续需要数据时再进行数据提取的方法，这样的设计可以减小内存占用，也提高了框架的运行效率。

具体实现方面，APK 包预处理模块使用 Python 的 os 库实现对命令行指令的调用，然后利用 Android SDK 中自带的命令行工具 aapt 对 APK 包进行解析，获取指定的数据项。

3.3.2 迭代搜索器

这个组件的设计利用了一个基于广度优先搜索（Breadth-First Search，简称 BFS）的算法，详情可见算法 3.1。根据缓存中的正版应用信息，本组件向应用收集器提交查询、下载样本的请求。在利用应用过滤器过滤获得的应用信息后，再根据其中正版应用的信息扩增缓存中的数据，进行下一轮迭代搜索。

算法 3.1 的输入有两项，分别是已有的正版应用信息列表 *legalAppInfo* 和要用来拓展搜索范围的数据项 *targetItems*。

算法开始前，本框架先遍历每个正版应用的信息，将每个要拓展搜索的数据项对应的内容 *apkInfo[item]* 插入到待查询队列 *wtQueue* 中，完成初始化（第 4 - 6 行）。然后是迭代搜索应用。每次迭代中，本算法都从 *wtQueue* 中取出一组键值对，其中键 *key* 为本次迭代中用于搜索新应用的数据项，*val* 为数据项对应的值（第 8 行）。取出键值对之后，算法先检查该键值对是否已经被用于之前的搜索中。如果该键值对之前已经出现过，就跳过本轮迭代，重新取一组键值对进行搜索；否则，将本组键值对放入数据缓存 *cache*，表示该组键值对已被使用过。之后，组件将键值对传递给应用收集器，应用收集器会生成对应查询，从应用来源中获取数据项相关的应用（第 12 行）。对于应用收集器返回的应用集 *newSamples* 中的每个样本 *sample*，本文会用应用过滤器检查 *sample* 是否为正版样本。如果是正版样本，那么本算法将从该样本中获取对应的数据 *sampleInfo*，然后将其中与待拓展对应项 *item* 对应的内容插入到待查询队列 *wtQueue* 中（第 15 - 18 行）。当应用集 *newSamples* 中的所有样本都被筛选检查过之后，本轮迭代结束。如果 *wtQueue* 中

算法 3.1: 迭代搜索算法

```

1  输入: targetItems, 列表, 用于拓展的数据项
2  输入: legalApkInfo, 列表, 包含正版应用及其信息键值对
3  输出: cache, 列表, 缓存拓展后的正版应用信息键值对

1 Function iterSearcher(legalApkInfo, targetItems):
2     wtQueue =  $\emptyset$ ;
3     cache =  $\emptyset$ ;
4     for apkInfo  $\in$  legalApkInfo do
5         for item  $\in$  targetItems do
6             wtQueue.add(item : apkInfo[item]);
7     while wtQueue  $\neq \emptyset$  do
8         key, val = wtQueue.pop();
9         if key : val  $\in$  cache then
10            continue;
11            cache.add(key : val);
12            newSamples  $\leftarrow$  appRetriever(key, val);
13            for sample  $\in$  newSamples do
14                isLegal  $\leftarrow$  FakeFilter(sample).getResult();
15                if isLegal then
16                    sampleInfo  $\leftarrow$  sample.getInfo();
17                    for item  $\in$  targetItems do
18                        wtQueue.add(item : sampleInfo[item]);
19    return cache;

```

的所有键值对都已经被检索完毕（即 `wtQueue` 为空），本算法流程结束，本组件会将 `cache` 中被用于拓展搜索的各个数据项键值对整理成 JSON 文件输出，方便之后的再利用。

在本次实证研究场景中, *targetItems* 包含两项内容, 一个是应用的包名 (*PackageName*), 另一个是应用自身的名字 (*AppName*)。之所以要这样操作, 是因为开发者推出的 App 的应用名和包名并不是一成不变的。一些热门应用会出于商业原因频繁地更改自己的应用名 (比如爱奇艺视频, 会根据其近期热播的电视剧/电影变更其应用名以吸引更多用户使用); 也有个别的热门应用可能会更换自己的包名, 比如 App 有重大改版、又或者是开发者安全证书有变更, 开发者不得不更换包名。

(具体原因可参考第 2.1 节的 Android App 签名机制部分)。

3.3.3 应用过滤器

顾名思义，应用过滤器的功能是从输入的应用程序集之中将仿冒应用筛选出来，其核心是安全证书的识别。根据第 2.1 节中对 Android 应用签名机制的描述，一个 App 中包含的安全证书文件指示了对 APK 文件进行修改的开发者。如果一个 APK 文件中包含的安全证书信息与指定应用的开发者信息相符，那么本组件认为这个 APK 包来源于正版的开发者；否则本组件认为这是一个仿冒应用。

在开始过滤之前，开发者需要先向过滤器导入正版证书信息。之后，对于每个输入的应用，组件会将其安全证书信息和正版证书信息作比对。对于证书信息相符的应用，本组件会将其放入迭代搜索器的待检索队列中用作下一轮的迭代搜索；而证书信息不相符的应用则会被筛选出并保存。

在迭代搜索的流程结束之后，本组件会将保存的所有仿冒应用数据项导出成 CSV 文件保存，方便之后的数据挖掘。

3.4 仿冒应用数据概览

以下是框架收集到的数据的概览：

从易观千帆提供的数据榜单中，本研究选择了 50 个最热门的 App 作为目标 App，这些 App 分属 11 个不同的应用类别。由于 App 的应用名可能会在 App 更新迭代的时候随之变更，笔者用基于 BFS 的策略，从 50 种 App 中一共记录了 198 个不同的应用名，来挖掘仿冒样本。

在这 50 款 App 中，以下三款 App 的样本并不能在市面上找到：OPPO 应用商店，华为应用商店和小米应用商店。因为这三款 App 都是由手机设备厂商开发和预装在对应品牌的手机中的，仅供这些品牌的用户使用，并不在其他应用市场上提供下载。当然，这也是这三款 App 热度高的原因——这几款 App 都被预装到了对应手机品牌厂商的每一部 Android 设备中，而 OPPO、华为和小米又是国内最大的几家手机厂商，这几款 App 自然也会有庞大的用户基数。因此，本研究最后的目标 App 只有 47 款。

对这 47 款目标 App，笔者总共收集到了 138,106 个应用样本。其中，69,614 个应用样本持有官方开发者证书，52,638 个应用样本并不具有官方证书。还有一部分应用样本，是某些应用的分别发布在不同应用市场同一版本，在经过去重筛选后被排除（共计 15,854 个）。

对于每个样本，框架收集 8 个数据项作为元数据：样本 *SHA1* 码，安全证书 *SHA1* 码，包名，样本大小，版本号，搜集时间和 *APK* 包来源。其中，样本 *SHA1* 码是使用 *SHA1* 哈希算法对整个 *APK* 文件进行数据摘要之后获取到的编码串，每个样本都有独一无二的 *SHA1* 码；安全证书 *SHA1* 码则是对样本的安全证书采用 *SHA1* 算法提取数据摘要之后获取的编码串，用于识别不同的证书。而搜集时间则是样本从应用市场被爬取到数据库的时间点，*APK* 包来源指示该 *APK* 包来源的应用市场。

3.5 本章小结

本章介绍了数据收集的流程和方法，然后讲述了面向仿冒应用的收集框架 *FakeRevealer* 的工作流程，以及其中三个组件——应用收集器、迭代搜索器和应用过滤器的设计实现，最后对采集到的数据进行了简要描绘。

尽管 *FakeRevealer* 在设计之时选择了利用包名和应用名迭代搜索、通过安全证书筛选的机制过滤仿冒应用，但框架本身的流程并不囿于此机制中，读者可以参考本框架流程，设计其他过滤仿冒应用，或者是其他任何具有某种特征的应用的工具。

在明确数据收集流程之后，本研究针对仿冒样本中采集到的元数据进行了挖掘分析，以求获得对于仿冒应用生态和特征、以及对于仿冒应用开发者的行为的更全面的认知。

第四章 结合实例分析的仿冒应用特征解读

4.1 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。

4.2 仿冒应用的基本特征

为了了解仿冒作者使用的仿冒策略，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，除非仿冒作者有意为之。

4.2.1 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 4.1 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系。

表 4.1: 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 节 Android App 签名机制最后的说明，笔者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。相关内容会在第??章中加以拓展。

4.2.2 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，笔者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，笔者找到了 14,089 个不同的包名。这其实在笔者的意料之中。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^[45]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”替换作“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。

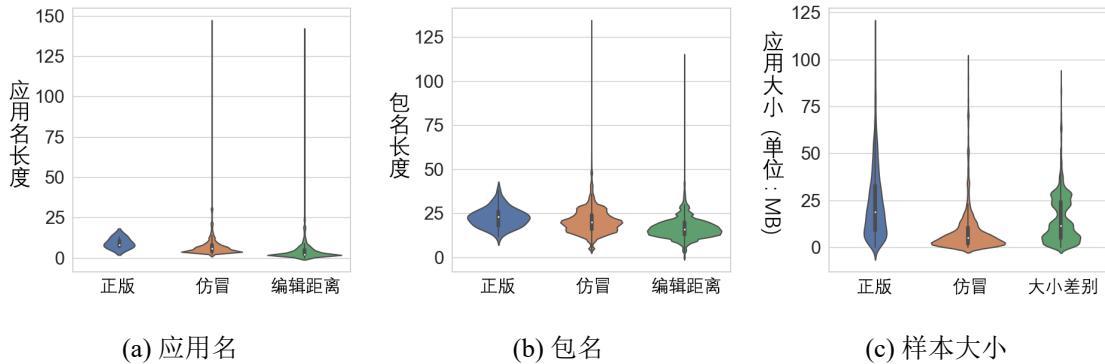


图 4.1: 对 App 各项属性的统计结果

图 4.1由三个小提琴图^[46]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。

4.1a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，笔者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）。笔者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。

4.1b显示了针对包名的结果。和 4.1a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。

4.1c 显示了 APK 包大小的信息。为了能更好地显示结果，笔者作图前从数据集中剔除了一些极端样本：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 4.1 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，笔者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 4.2 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 4.2d 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。

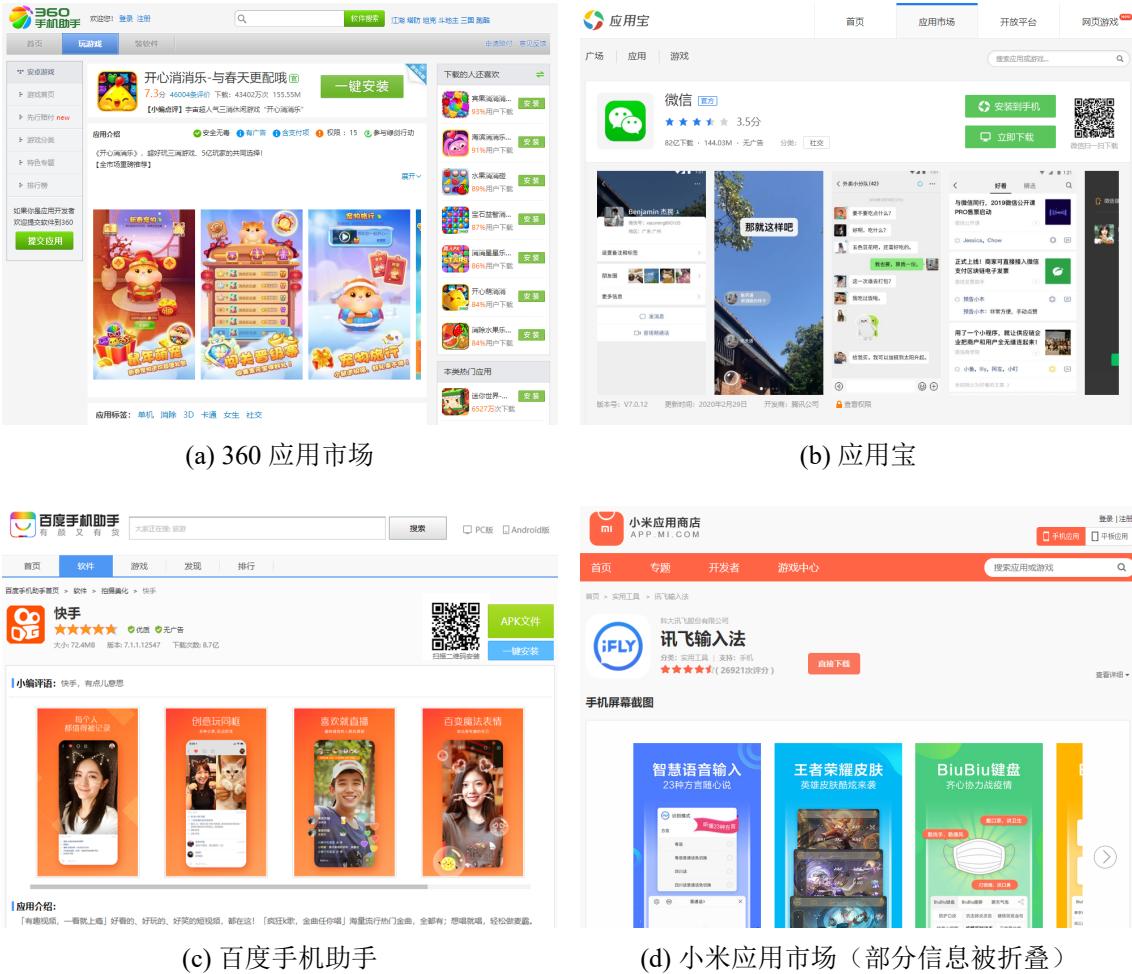


图 4.2: 各大应用市场应用详情页 (从桌面端浏览)

4.2.3 案例 1. 持有官方安全证书的可疑样本

在人工浏览数据时，笔者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。于是，笔者针对所有 69,614 个持有官方证书的样本进行了以下几项筛选：

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 4.2 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，笔者发现该证书来自 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，所以笔者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 4.2: 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，笔者起初不禁怀疑是否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）^[47]。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 版本签名机制的漏洞，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。

4.3 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。

4.3.1 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。因此笔者不禁好奇，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 4.3 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 31 个应用商店中，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。

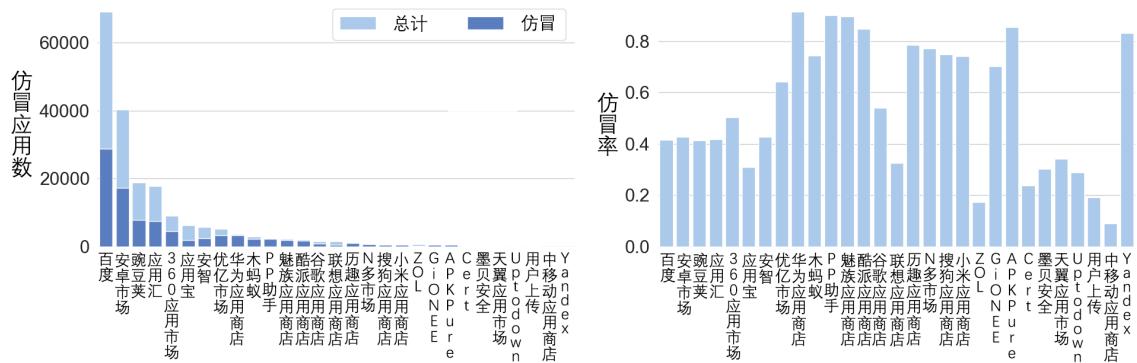


图 4.3: 从不同应用市场中收集到的应用数量以及各市场仿冒率

定义 2 仿冒率

某款 App a 的仿冒率 $fake\ sample\ rate_a$ 指与其关联的仿冒样本的数量 $fake_a$ 与该 App 正版样本的数量 $total_a$ 的商（式 4.1）。

某应用市场 AS 的仿冒率 $fake\ sample\ rate_{AS}$ 则是其中包含的所有目标 App 的均值（式 4.2）。

$$fake\ sample\ rate_a = \frac{fake_a}{total_a} \quad (4.1)$$

$$fake\ sample\ rate_{AS} = Avg(fake\ sample\ rate_a, \forall a \in \{\text{目标 App}\}) \quad (4.2)$$

尽管百度手机助手^[40]和安卓市场^[48]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^[39]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。

4.3.2 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 4.1 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^[43] 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 4.3）。

$$p_{x,y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} = \frac{E[(X - u_x)(Y - u_y)]}{\sigma_x \sigma_y} \quad (4.3)$$

式 4.3 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。出乎笔者意料的是，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，和笔者的预期并不符合。

RQ 2.3. 结果 笔者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，笔者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。笔者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^[49] 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 4.3: 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
开心消消乐*	游戏	11.2K	19.7	2406	1738	72.2%	20.6
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
WPS Office*	商务办公	7.4K	6.0	1152	69	6.0%	7.8
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
QQ 邮箱*	商务办公	6.6K	16.4	520	11	2.1%	10.4

* 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 4.3 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分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。

这个结果与笔者在日常生活中观察到的结果相符，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。

4.3.3 案例 2. 游戏类别下的仿冒应用

正如表 4.3 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，笔者随机从这两款游戏 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。

4.4a 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 4.4: 游戏类 App 及其仿冒样本

笔者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。笔者使用的测试设备为高配版小米 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)。

4.4b 和 4.4c 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。笔者在

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，连笔者都没有办法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，笔者发现 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^[50] 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。

笔者认为，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 4.3 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。笔者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。

4.4 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。

4.4.1 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：

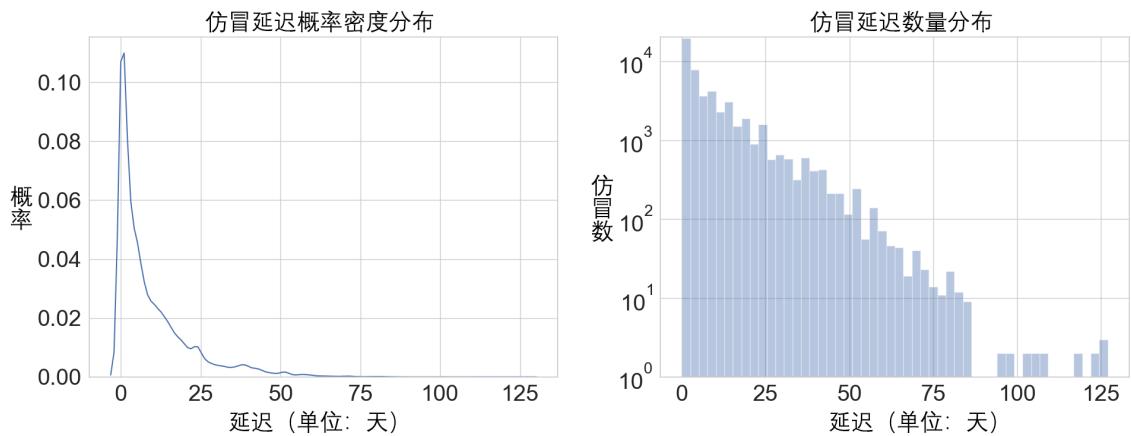


图 4.5: 仿冒延迟总体分布

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 4.5 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 中每个版本最早的发布日期串联起来，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 4.5 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。

4.4.2 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 4.6 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 4.6 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 4.6 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

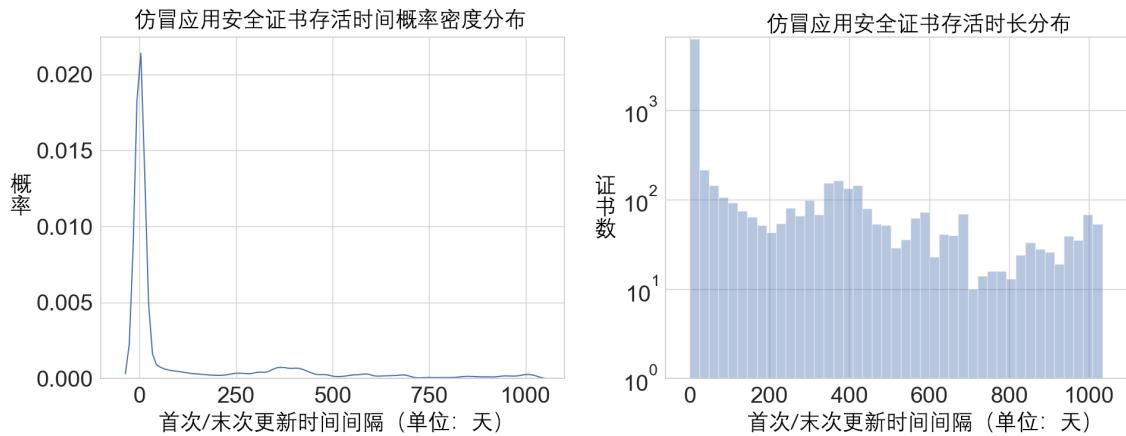


图 4.6: 仿冒应用安全证书存活时间分布

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，笔者也能发现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。

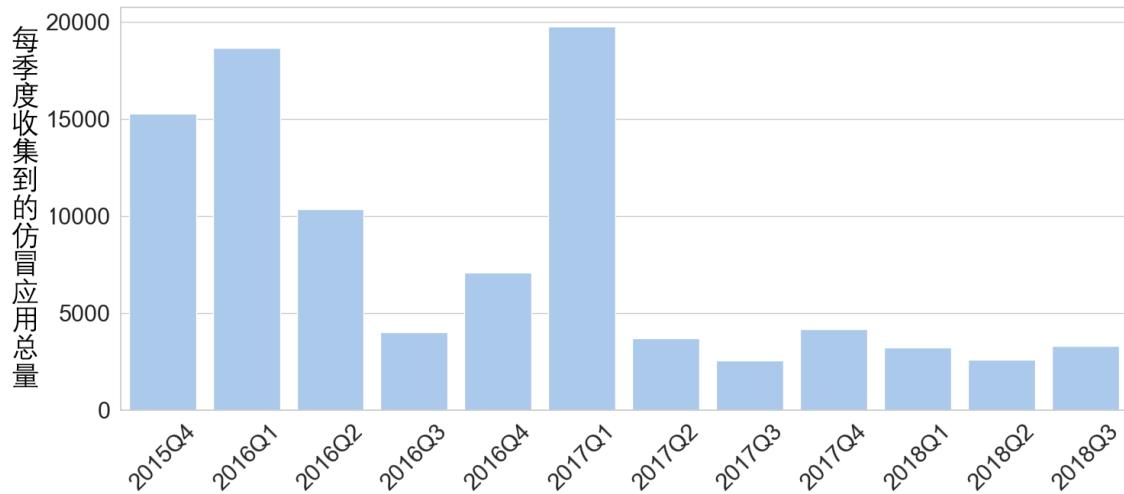


图 4.7: 每季度爬取到的仿冒样本数量（2015 年第四季度到 2018 年第三季度）

4.4.3 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？

笔者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 4.7 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^[51]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^[52]。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^[53]。所以笔者有理由猜测 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。

4.4.4 案例 3. 与大量仿冒应用相关联的仿冒应用安全证书

第 4.2 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，笔者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，笔者却发现了意料之外的结果。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 4.4: 由“61ed377e85d386a8dfee6b864bd85b0bfaa5af81”签署的部分样本

Name	Package Name	Size
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.tencent.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 4.4 中展现了由这个安全证书签署的部分样本。笔者将这些样本上传到知名在线反病毒引擎 VIRUSTOTAL^[50] 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，笔者推导，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。

4.5 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。

第五章 面向仿冒应用的排名欺诈验证方法

5.1 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^[54] 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^[29] 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。

5.2 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，所以笔者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，笔者随机挑选了 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，笔者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，笔者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。

5.2.1 评论数据概览

在笔者收集评论的所有 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 App 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，笔者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。

5.2.2 基础分析

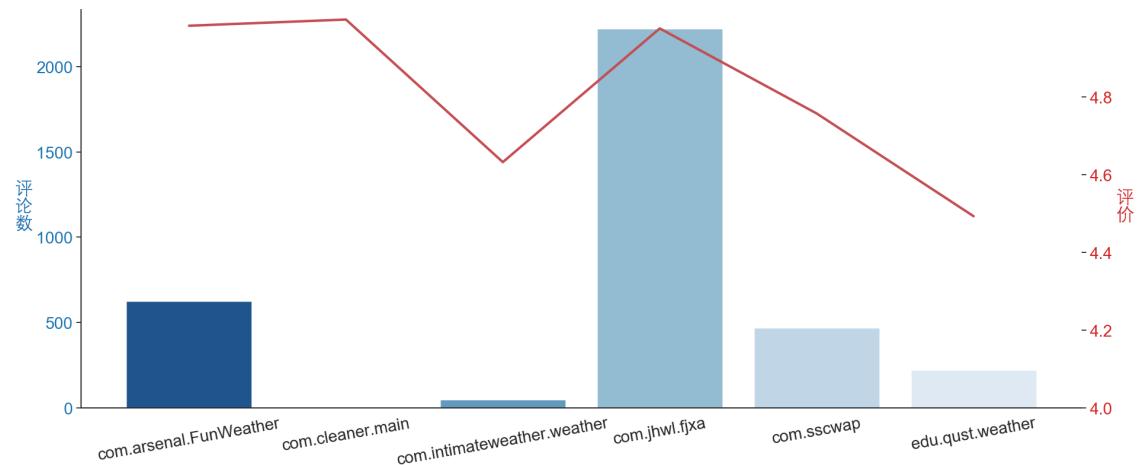


图 5.1: 各仿冒应用在 360 手机助手商店中的评论数量与评分分布

图 5.1 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.jhwl.fjxa* 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.sscswap* 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笔者想到恶意刷评的相关研究。然而，笔者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 5.2 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。

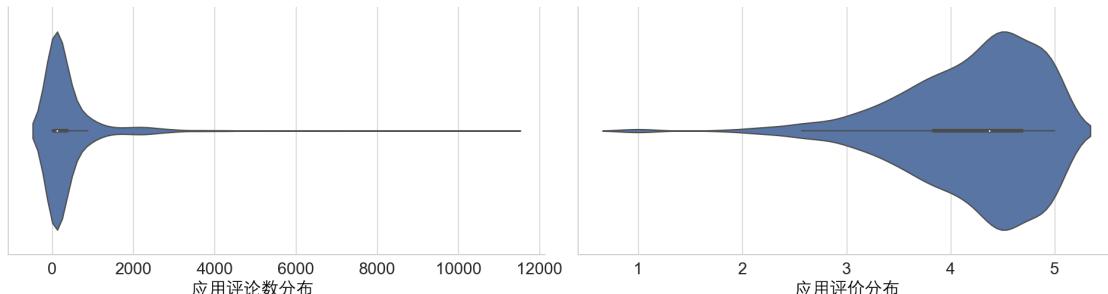


图 5.2: 所有 856 个应用在商店中的评论数量与评分分布

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.sscswap*。

结合两个维度的分布结果，*com.jhwl.fjxa*、*com.arsenal.FunWeather*、*com.sscswap* 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.sscswap* 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，因此需要验证上述应用是否利用了排名欺诈将评分拉高。

5.3 仿冒应用与排名欺诈关联验证

5.3.1 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，所以本研究选择了另一前人研究的方法进行探索。**FRAUDAR**^[33] 是 Bryan Hooi 在 2016 年推出的一个算法，可以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%。笔者将 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，笔者发现，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）。为确认 FRAUDAR 的有效性，本研究提出了另一种排查方法。

5.3.2 基于评论用户可信度的排名欺诈排查

Mohammad-Ali^[55] 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 5.1 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 5.1 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 5.2 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。

$$Sim(u_i, u_j) = \frac{1}{t_n - t_0} \sum_{t=t_0}^{t_n} \sigma(B(u_i, t), B(u_j, t)) \quad (5.1)$$

$$Jaccard(set_i, set_j) = \frac{|set_i \cap set_j|}{|set_i \cup set_j|} \quad (5.2)$$

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 5.3 中的二元组表示：

$$W_{app_k} = (w_{app_k}^0, w_{app_k}^1) \quad (5.3)$$

$$w_{app_k}^0 = \frac{|\{u_i \mid u_i \in G_r, cmt_j \rightarrow u_i, cmt_j \in CL_{app_k}\}|}{|G_r|} \quad (5.4)$$

$$w_{app_k}^1 = \frac{|\{cmt_j \mid cmt_j \in CL_{app_k}, cmt_j \rightarrow u_i, u_i \in G_r\}|}{|CL_{app_k}|} \quad (5.5)$$

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。

笔者分别将 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，笔者却发现这些聚类分别包含了 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。因此笔者推定，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，笔者分别计算了不同 T_1 下，三款仿冒应用在式 5.4 和式 5.5 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 5.1 和表 5.2 中。

表 5.1: 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	5×10^{-3}	346	5×10^{-3}	318

表 5.2: 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	2×10^{-4}	261	8×10^{-5}	279	9×10^{-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。

5.3.3 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。根据笔者的经验，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。

$$RD_{app_k} = 1 - \frac{\sum_{i=0}^n |CG_i|}{|CL_{app_k}|} \quad (5.6)$$

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¹，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 5.2 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。

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 5.3，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

¹<https://github.com/fxsjy/jieba>

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。

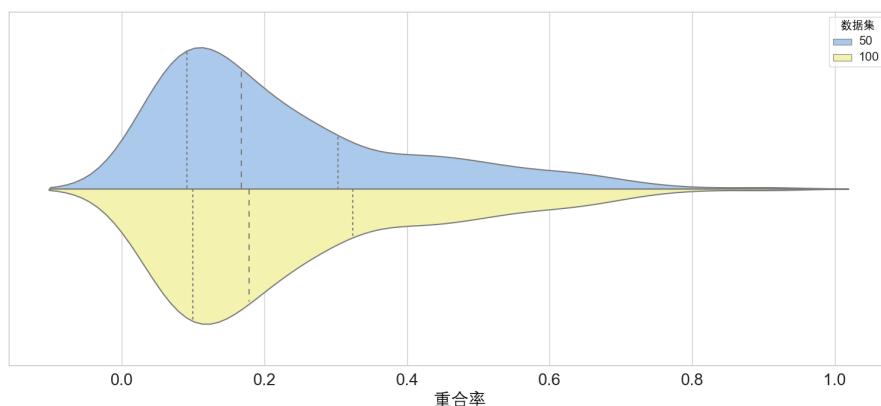


图 5.3: 两种数据组的应用评论重合率结果

表 5.3: 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 5.3 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%), 都算是比较靠前的位置。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。

5.3.4 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，笔者对每个应用的评论都按内容进行了排序。

图 5.4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，笔者不倾向于认为这款 App 购买了排名欺诈服务。反观 5.4a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 5.4b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。

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。

5.4 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

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 5.4: 应用评论取样

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。

第六章 总结与展望

6.1 总结

在本文中，笔者率先引入了“仿冒应用”这个概念，然后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。据笔者所知，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超过 15 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，笔者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，呈现了如仿冒作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。

笔者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。

6.2 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:** 第四章中提及到了 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 App 进行攻击，其实也可以降低目标 App 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。笔者希望本文能抛砖引玉，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] 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.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] CHEN S, XUE M, FAN L, et al. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach[J]. computers & security, 2018, 73 : 326–344.

- [9] CHEN S, XUE M, XU L. Towards adversarial detection of mobile malware: poster[C] // Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking. 2016 : 415–416.
- [10] FAN L, XUE M, CHEN S, et al. POSTER: Accuracy vs. Time Cost: Detecting Android Malware through Pareto Ensemble Pruning[C] // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016 : 1748–1750.
- [11] MCAFEE. McAfee Mobile Threat Report Q1, 2018[R/OL]. 2018 [September 26, 2018].
<https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf>.
- [12] YIN H, SONG D, EGELE M, et al. Panorama: capturing system-wide information flow for malware detection and analysis[C] // Proceedings of the 14th ACM conference on Computer and communications security. 2007 : 116–127.
- [13] FELT A P, FINIFTER M, CHIN E, et al. A survey of mobile malware in the wild[C] // SPSM@CCS. 2011.
- [14] ZHOU Y, JIANG X. Dissecting Android Malware: Characterization and Evolution[J]. 2012 IEEE Symposium on Security and Privacy, 2012 : 95–109.
- [15] 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.
- [16] ZHOU W, ZHOU Y, JIANG X, et al. Detecting repackaged smartphone applications in third-party android marketplaces[C] // CODASPY. 2012.
- [17] ZHENG M, SUN M, LUI J C S. DroidAnalytics : A Signature Based Analytic System to Collect , Extract , Analyze and Associate Android[C] // . 2013.
- [18] 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.

- [19] GLANZ L, AMANN S, EICHBERG 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.
- [20] 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.
- [21] ZHANG F, HUANG H, ZHU S, et al. ViewDroid: towards obfuscation-resilient mobile application repackaging detection[C] // WISEC. 2014.
- [22] 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.
- [23] CRUSSELL J, GIBLER C, CHEN H. Scalable semantics-based detection of similar Android applications[C] // Proc. of ESORICS : Vol 13. 2013.
- [24] 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.
- [25] FU B, LIN J, LI L, et al. Why people hate your app: Making sense of user feedback in a mobile app store[C] // Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. 2013 : 1276 – 1284.
- [26] KONG D, CEN L, JIN H. Autoreb: Automatically understanding the review-to-behavior fidelity in android applications[C] // Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015 : 530 – 541.
- [27] HERNANDEZ N, RAHMAN M, RECABARREN R, et al. The Art and Craft of Fraudulent App Promotion in Google Play[C] // Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS 2019). 2019.
- [28] 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.

- [29] 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.
- [30] HU Y, WANG H, LI L, et al. Want to earn a few extra bucks? a first look at money-making apps[C] // 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2019: 332–343.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] Janus[EB/OL]. 2020 [March 27, 2020].
<https://www.apbscan.io/>.
- [36] Anjian[EB/OL]. 2020 [March 20, 2020].
<http://m.anjian.com/>.
- [37] iApp[EB/OL]. 2020 [March 20, 2020].
<http://yougais.com/>.
- [38] MCAFEE. 黑灰产的廉价“温床”——APP 生成框架 [R/OL]. 2018 [May 16, 2019].
<https://www.anquanke.com/post/id/178540>.

- [39] Myapp[EB/OL]. [September 26, 2018].
<http://sj.qq.com/myapp/>.
- [40] Baidu App Store[EB/OL]. [September 26, 2018].
<https://shouji.baidu.com/>.
- [41] Xiaomi App Store[EB/OL]. [February 23, 2020].
<http://app.mi.com/>.
- [42] PANDITA R, XIAO X, YANG W, et al. {WHYPER}: Towards Automating Risk Assessment of Mobile Applications[C] // Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13). 2013 : 527–542.
- [43] Analysys Figure[EB/OL]. [September 26, 2018].
<https://qianfan.analysys.cn/refine/view/rankApp/rankApp.html>.
- [44] Burpsuite[EB/OL]. 2020 [March 21, 2020].
<https://portswigger.net/>.
- [45] LEVENSHTEIN V I. Binary codes capable of correcting deletions, insertions, and reversals[C] // Soviet physics doklady : Vol 10. 1966 : 707–710.
- [46] Violin plot[EB/OL]. 2020 [March 27, 2020].
https://en.wikipedia.org/wiki/Violin_plot.
- [47] GOOGLE. Android Security Bulletin—December 2017[EB/OL]. 2017 [September 26, 2018].
<https://source.android.com/security/bulletin/2017-12-01>.
- [48] Hiapk[EB/OL]. [September 26, 2018].
<http://apk.hiapk.com/>.
- [49] 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.
- [50] VirusTotal[EB/OL]. [September 26, 2018].
<https://www.virustotal.com/>.

- [51] WIKIPEDIA. Ransomware[EB/OL]. 2018 [September 26, 2018].
<https://en.wikipedia.org/wiki/Ransomware>.
- [52] 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/>.
- [53] 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/>.
- [54] 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.
- [55] 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. Chongbin Tang, Sen Chen, Lingling Fan, Lihua Xu, Yang Liu, Zhushou Tang, and Liang Dou, “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. (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