10-31-2018

Search Rank Fraud Prevention in Online Systems

Md Mizanur Rahman
mrahm031@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SEARCH RANK FRAUD PREVENTION IN ONLINE SYSTEMS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Md Mizanur Rahman

2018

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Md Mizanur Rahman, and entitled Search Rank Fraud Prevention in Online Systems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Geoffrey Smith

Shu-Ching Chen

Mark Finlayson

Selcuk Uluagac

Bogdan Carbunar, Major Professor

Date of Defense: October 31, 2018

The dissertation of Md Mizanur Rahman is approved.

Dean John L. Volakis
College of Engineering and Computing

Andres G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2018

DEDICATION

To my parents Abdur Rahim, Monowara Begum and my loving wife Nowreen

ACKNOWLEDGMENTS

First and foremost, I would like to express my deep and sincere gratitude to my supervisor Dr. Bogdan Carbunar for his inspiration, patience, and encouragement. He introduced me to the field of Security and Machine Learning and provided lots of good ideas and advice. His guidance helped me in all the time of research and writing of this thesis. He consistently helped me and steered me in the right direction whenever he thought I needed it. I would have never completed this thesis without his help. I could not have imagined having a better advisor and mentor for my Ph.D. study.

I would also like to thank the co-authors of my papers: Dr. Duen Horng Chau, Dr. Dongwon Lee, Dr. Rahul Potharaju and Dr. Mahmudur Rahman for their encouragement, support, and guidance. They provided me an opportunity to work with them and without their precious support, it would not be possible to conduct this research.

I am so grateful to the Faculty of School of Computing and Information Sciences at Florida International University for making it possible for me to study here. I give deep thanks to the Professors, lecturers and other workers of the faculty. Additionally, I want to thank the rest of my committee: Dr. Geoffrey Smith, Dr. Shu-Ching Chen, Dr. Mark Finlayson and Dr. Selcuk Uluagac for agreeing to serve on my committee and for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives.

I thank my fellow labmates Nestor, Ruben, Mozhgan, and Sajib for making the CaSPR lab awesome, being fun to work with and everything in between. Both Nestor and Ruben are collaborators on my several papers and, through working together over the last several years, each has become one of my very dearest friends. I am looking forward to each of them putting up with lots more of me in the future.

A very special gratitude goes out to all down at National Science Foundation (NSF), FIU SCIS, Graduate School and the Florida Center for Cybersecurity for helping and providing the funding for the work. This dissertation would not have been possible without funding from these funding agencies. This research was supported by the NSF under grant CNS-1527153, CNS-1526254, CNS-1526494, SES-1450619, CNS-1422215, IUSE-1525601. This research was supported in part by DoD grant W911NF-13-1-0142 and Samsung GRO. Also, I would like to thank Graduate School to grant Dissertation Year Fellowship for which I am grateful.

Last but not the least, I would like to thank my family: my parents Abdur Rahim, Monowara and to my brother Mustafiz and sister Renu for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Most importantly, I wish to thank my loving and supportive wife, Nowreen who has given everything possible and even given up important things to make sure I achieve this feat. I can not find the words that express my gratitude. This accomplishment would not have been possible without them. Thank you.

ABSTRACT OF THE DISSERTATION

SEARCH RANK FRAUD PREVENTION IN ONLINE SYSTEMS

by

Md Mizanur Rahman

Florida International University, 2018

Miami, Florida

Professor Bogdan Carbunar, Major Professor

The survival of products in online services such as Google Play, Yelp, Facebook and Amazon, is contingent on their search rank. This, along with the social impact of such services, has also turned them into a lucrative medium for fraudulently influencing public opinion. Motivated by the need to aggressively promote products, communities that specialize in social network fraud (e.g., fake opinions and reviews, likes, followers, app installs) have emerged, to create a black market for fraudulent search optimization. Fraudulent product developers exploit these communities to hire teams of workers willing and able to commit fraud collectively, emulating realistic, spontaneous activities from unrelated people. We call this behavior search rank fraud.

In this thesis, we introduce two novel approaches to discourage search rank fraud in online systems. First, we detect fraud in real-time, when it is posted, and impose resource consuming penalties on the devices that post activities. We introduce and leverage several novel concepts that include (i) stateless, verifiable computational puzzles that impose minimal performance overhead, but enable the efficient verification of their authenticity, (ii) a real-time, graph based solution to assign fraud scores to user activities, and (iii) mechanisms to dynamically adjust puzzle difficulty levels based on fraud scores and the computational capabilities of devices.

In a second approach, we introduce the problem of fraud de-anonymization: reveal the crowdsourcing site accounts of the people who post large amounts of fraud, thus their bank accounts, and provide compelling evidence of fraud to the users of products

that they promote. We investigate the ability of our solutions to ensure that fraud does not pay off. We survey the assumptions made by the fraud detection research community on the capabilities and behaviors of fraudsters, and study their relevance to app search optimization fraud conducted for Google Play apps. For this, we designed and conducted in-depth interviews with expert fraudsters recruited from several freelancing sites.

Our idea that fraud needs to be proactively discouraged and prevented, instead of only reactively detected and filtered, has the potential to evolutionize fraud detection for online, peer-review systems. Our techniques will help provide counter-incentives for fraud workers and the developers who hire them: Online systems will be able to expose influential fraud workers through their bank accounts, and provide compelling evidence of fraud to the users of products that they promote.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

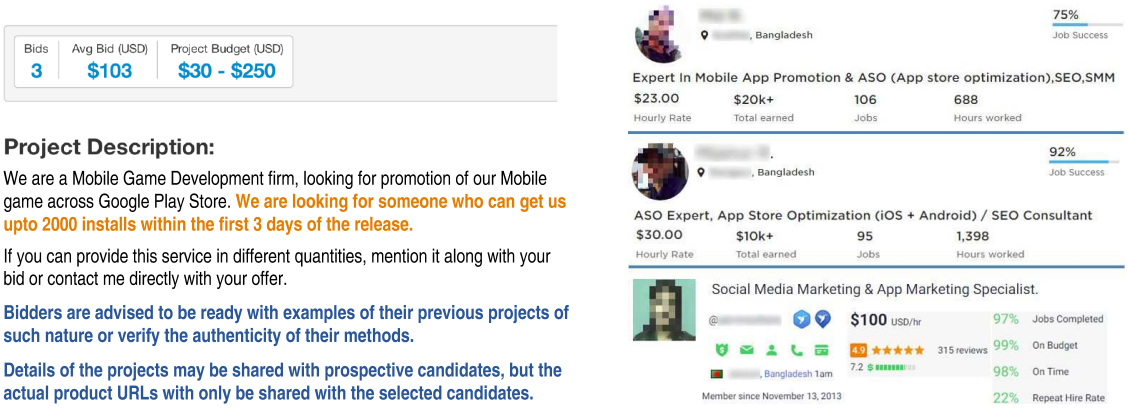
Introduction

1.1 Background

The social impact of online services built on information posted by their users[Goob, Yel, Fac, Twi] has also turned them into a lucrative medium for fraudulently influencing public opinion. For example, search rank fraud on sites like Google Play[Goob], Yelp[Yel] and Facebook[Fac]. People often assume the popularity of featured products is generated by purchases, downloads and reviews of real patrons, who are sharing their honest opinions about what they have experienced. Every day, people rely on online information to make decisions on purchases, services, software and opinions.

Motivated by the need to aggressively promote disinformation, communities that specialize in social network fraud (e.g., fake opinions and reviews, likes, followers, app installs) have emerged. The survival of mobile apps in markets such as Google Play is contingent on their search rank. Higher ranked apps are installed more frequently and generate more revenue, through ads or direct payments.

Popular belief (e.g., Ank (2013)) holds that large numbers of positive reviews help new apps achieve higher search rank. The commercial success of Android app markets such as Google Play and the incentive model they offer to popular apps, make them appealing targets for fraudulent and malicious behaviors. The resulting need of developers to aggressively promote their apps has created a black market for fraudulent app search optimization (ASO). Malicious developers use app markets as a launch pad for their malware[Min14, Mlo14, Rob15, Gre14]. In addition, the efforts of Android markets to identify and remove malware are not always successful. For instance, Google Play uses the Bouncer system[OM12] to remove malware. However, out of the 7, 756 Google Play apps we analyzed using VirusTotal[Vir15], 12% (948)

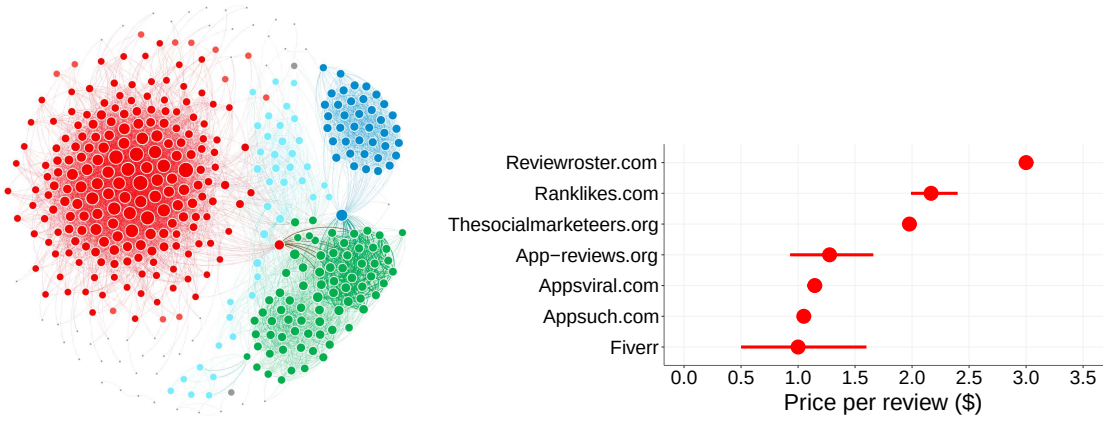


(a) (b)

Figure 1.1: (a) An install job posting from Freelancer [Fre], asking for 2000 installs within 3 days (in orange), in an organized way that includes expertise verifications and provides secrecy assurances (in blue). Text enlarged for easier reading. (b) Anonymized snapshots of profiles of search rank fraudsters from Upwork (top 2) and Freelancer (bottom). Fraudsters control hundreds of user accounts and earn thousands of dollars through hundreds of work hours. Our goal is to de-anonymize fraud, i.e., attribute fraud detected for products in online systems, to the crowdsourcing site accounts of the fraudsters (such as these) who posted it.

were flagged by at least one anti-virus tool and 2% (150) were identified as malware by at least 10 tools.

Crowdsourcing sites (e.g.,Fiverr, Upwork, Freelancer)[Upw, Fiv, Fre]) play a central role in this ecosystem. While general purpose crowdsourcing sites enable fraudsters to advertise their services and match them with interested clients, specialized, fraud-as-a-service sites have emerged [TSM16, RR16, RL16, AV16, AS16, AR16] that focus solely on fraud work. Fraudulent developers frequently exploit crowdsourcing sites to hire teams of willing workers to commit fraud collectively, emulating realistic, spontaneous activities from unrelated people (i.e., "crowdturfing"[WWZ+12]). We call this behavior search rank fraud, see Figure 1.1(a) for an example. Fraudsters create hundreds of user accounts, connect with product developers through crowdsourcing sites, then post fake activities for their products, from the accounts they control, see Figure 1.1(b)



(a) (b)

Figure 1.2: (a) Co-activity graph of user accounts reviewing a popular horoscope app in Google Play (name hidden for privacy). Nodes are accounts. 4 Upwork workers revealed to control the accounts of the same color. Two accounts are connected if they post activities for similar sets of apps. Node sizes are a function of the account connectivity. (b) Price per review (minimum, average and maximum), for crowdsourcing sites that focus on app market fraud. The sites offer "fraud packages" and even discounts for bulk fake review purchases. A fake review costs between \$0.5-\$3.

This vibrant black market for online fraud signals profitability for fraudsters. Figure 1.2(b) shows that the competition pushes prices down, as low as \$1 per fake review. This further suggests that existing techniques (e.g., [YA15a, MLG12, HSB+16b, RRCC16b]) that flag fake reviews posted by surrogate accounts controlled by human fraudsters may be insufficient in curbing organized fraud.

Attempts to detect and filter fraud, e.g., [SCM11, Cal16] are known to be ineffective [RRCC16b, Sin15, Sie14, Mlo14, Min14]. For instance, Figure 1.2(a) shows the co-review graph of a Google Play Store app that received fake reviews from 4 workers in Freelancer [Fre]. Nodes with the same color denote reviewer accounts controlled by the same worker, while edges connect accounts with a history of review activities targeting the same apps.

1.2 System Model

1.2.1 Background: Operation of Online Services

The central component in our model is an online service (the *service*) that hosts the system. The service stores information about user accounts and featured *subjects*. Subjects can be mobile apps in stores such as Google Play [Goob], businesses in geosocial networks such as Yelp [Yel] and product items in eBay [eBa].

Users register with the service, record profile information (e.g., name, gender, home city) and receive initial service credentials, including a unique user id. Users can query the online service for information concerning subjects and other users. In popular social networks, such as Facebook and Twitter, the subjects can also include the user accounts and the information that they post.

Users are encouraged to *act* on existing subjects. The actions include posting reviews, comments, or likes, installing mobile app subjects, and following or friending user accounts. The online service associates statistics over the actions performed for each supported subject. To facilitate the retrieval of relevant subjects upon user query, the service ranks them according to their popularity. The action statistics maintained by the service have a significant impact on the popularity and search rank of subjects. Subjects that rank higher in search results tend to be more popular [Quo].

1.2.2 High Level Adversary Model

We consider that adversarial behaviors start with the subject *owners*. Adversarial owners seek to fraudulently promote their subjects, that can be of inferior quality and even malicious, or demote competitor subjects. The focus of this proposal will be on the *fraud workers* hired to launch such concerted efforts, or "fraud campaigns", to bias the popularity and public opinion of specific subjects.

Fraud campaigns can, for instance, make businesses and products more profitable [AM12, Luc], and increase the "reachability" of malware developers: more victims install their apps. Install campaigns seek to fraudulently increase the number of installs of an app. Review campaigns (sanctioned by the law [Att]) aim to increase the average rating of the subject.

While adversarial owners can be rational or purely malicious, fraud workers are rational, financially motivated adversaries. The jobs they receive specify a payment amount and a quota of fraud expected to be posted.

Fraud workers can specialize on specific online services and types of fraudulent actions, e.g., posting fake reviews, ratings or likes, installing apps, and following or friending other user accounts. In generic crowdsourcing sites like Upwork [Upw] and Freelancer [Fre], employers post jobs to hire fraud workers to target specific subjects. Workers can then **bid** on such jobs, and, following negotiation steps, be assigned, or **win** the jobs. In sites like Fiverr [Fiv], workers advertise their ability to perform ASO jobs; employers need to identify and hire workers with the required expertise.

In addition, numerous "fraud-as-a-service" (FAAS) sites exist, where clients buy fraud packages without direct interaction with the workers [TSM16, RR16, RL16, AV16, AS16, AR16]. Figure 1.2(b) shows the minimum, average and maximum cost per fraudulent action, as advertised by several FAAS sites: a fake review for an app can worth as low as $1, while a fake social networking "like" can be as low as $2.

1.3 Contributions

We observe that fraud detection, as proposed in academic work and implemented in popular online systems, is unable to ensure access to truthful information [Gro, Wei17, MEY17, TRU, BRO17, Sie14, Min14, Mlo14, Rob15, Gre14] In this thesis

we propose that **fraud needs to be proactively discouraged and prevented**, instead of only reactively detected and filtered.

We introduce two novel approaches to discourage search rank fraud in online systems. First, we seek to detect fraud in real-time, when it is posted, and to impose resource consuming penalties for posting fraud. Second, we propose the problem of fraud de-anonymization: reveal the crowdsourcing site accounts of the people who post large amounts of fraud, thus their bank accounts, and provide compelling evidence of fraud to users of products that they promote.

1.3.1 Longitudinal App Market Study

In preliminary work [PRC17], we performed a detailed longitudinal analysis on 87,223 newly released apps. We have developed GPCrawler, a tool to automatically collect data published by Google Play for apps, users and reviews. We have monitored and collected data about these apps over more than 6 months. Features extracted from a longitudinal app analysis (e.g., permission, price, update, download count changes) can provide insights into fraudulent app promotion and malware indicator behaviors. We found that a high number of apps have not been updated over the monitoring interval. Moreover, these apps were controlled by a few developers that dominate the total number of app downloads. At most 50% of the apps in each category have received an update within 35 days, while apps in the Social and Tools categories received updates within 15 days.

1.3.2 Fraud and Malware Detection in Google Play

We have developed FairPlay [RRCC16a], a system that discovered and leveraged traces left behind by fraudsters, to detect both malware and apps subjected to search rank fraud. We have correlated review activities and uniquely combines detected

review relations with linguistic and behavioral signals gleaned from Google Play app data (87K apps, 2.9M reviews, and 2.4M reviewers, collected over half a year), in order to identify suspicious apps.

FairPlay organized the analysis of longitudinal app data into the 4 modules. The Co-Review Graph (CoReG) module identified apps reviewed in a contiguous time window by groups of users with significantly overlapping review histories. The Review Feedback (RF) module exploited feedback left by genuine reviewers, while the Inter Review Relation (IRR) module leveraged relations between reviews, ratings and install counts. The Jekyll-Hyde (JH) module monitored app permissions, with a focus on dangerous ones. Each module generated several features that we used to train an app classifier. FairPlay also used general features such as the apps average rating, total number of reviews, ratings and installs, for a total of 28 features.

FairPlay achieved over 95% accuracy in classifying gold standard datasets of malware, fraudulent and legitimate apps. We show that 75% of the identified malware apps engaged in search rank fraud. FairPlay discovered hundreds of fraudulent apps that evaded Google Bouncers detection technology. FairPlay also enabled us to discover more than 1,000 reviews, reported for 193 apps, that reveal a new type of coercive review campaign: users are harassed into writing positive reviews, and install and review other apps.

1.3.3 Real-time Fraud Preemption System

We introduced the concept of *fraud preemption systems*, solutions deployed to defend online systems such as social networks and app markets. We proposed a real-time fraud preemption approach that imposes Bitcoin-inspired computational puzzles on the devices that post online system activities, such as reviews and likes. We have adapted computational puzzles to prevent online system fraud and to solve the addi-

tional challenges of building puzzles whose difficulty is a function of the probability that an activity is fraudulent while handling heterogeneous user devices (e.g., ranging from smartphones to machines that specialize in such puzzles).

We have developed FraudSys [RRCL17], a real-time fraud preemption approach by leveraging several novel concepts that include (i) stateless, veriable computational puzzles, that impose minimal performance overhead, but enable the ecient verication of their authenticity, (ii) a real-time, graph based solution to assign fraud scores to user activities, and (iii) mechanisms to dynamically adjust puzzle diculty levels based on fraud scores and the computational capabilities of devices.

To validate FraudSys, we: (1) quantified the computation penalty imposed on fraudsters; (2) introduced upper bounds on the profitability of fraud and the amount of fraud that can be created for a single subject, per time unit; (3) used two real datasets Google Play (i.e., 23K fake reviews from 23 crowdsourced fraud workers, and 1K honest reviews) and Facebook (i.e., 274K fake and 180K genuine Likes), to demonstrate that FraudSys imposes daily penalties as high as 3,079 hours on a single fraudster, and, for a properly equipped fraudster, makes fraud less productive than Bitcoin mining.

1.3.4 Fraud De-anonymization

We introduce the *fraud de-anonymization* problem, that goes beyond fraud detection, to unmask the human masterminds responsible for posting search rank fraud in online systems. We collect and study search rank fraud data from Upwork, and survey the capabilities and behaviors of 58 search rank fraudsters recruited from 6 crowdsourcing sites. We propose Dolos, a fraud de-anonymization system that leverages traits and behaviors extracted from these studies, to attribute detected fraud to crowdsourcing site fraudsters, thus to real identities and bank accounts. We introduce MCDense,

a min-cut dense component detection algorithm to uncover groups of user accounts controlled by *different* fraudsters, and leverage stylometry and deep learning to attribute them to crowdsourcing site profiles. Dolos correctly identified the owners of 95% of fraudster-controlled communities, and uncovered fraudsters who promoted as many as 97.5% of fraud apps we collected from Google Play. When evaluated on 13,087 apps (820,760 reviews), which we monitored over more than 6 months, Dolos identified 1,056 apps with suspicious reviewer groups. We report orthogonal evidence of their fraud, including fraud duplicates and fraud re-posts.

1.3.5 Crowdsourced Review Fraud Survey

we study the search rank fraud ecosystem that targets peer-opinion sites, in particular, Google Play. The diversity, responsiveness and revealed profitability of this ecosystem, suggest an inaccurate and incomplete understanding of the capabilities, behaviors and strategies of fraud workers, including their strategies to avoid detection. The goal is to determine which existing assumptions still hold, whether fraudsters have developed fraud detection strategies to avoid being identified by the peer-opinion site, to determine the capabilities, behaviors and strategies employed by fraudsters, and identify weakness that can be exploited by online systems and researchers, to identify and prevent fraud.

1.4 Fundamental Challenges

- **Multiple fraudster strategies**. We cannot assume that all the fraudsters employ the same search rank fraud posting strategy. Our preliminary work revealed that fraudsters have various resources and different levels of expertise, and use different strategies to post fraud. The strategies evolve along with the defense mechanisms implemented by online systems.

- **Lack of ground truth data**. Fraud detection and de-anonymization requires knowledge of at least a modicum of ground truth data, including not only datasets of fraudulent and honest activities, but also of attributed fraudulent activities: the crowdsourcing account id of the worker who posted a fraudulent activity. Collecting such data is hard and raises ethical issues, while quality data is practically inexistent in the literature.
- **Real-time fraud detection**. The real-time constraint of identifying fraud implies that fraud detection needs to achieve high accuracy given only information from the past, and no knowledge of the future. Once the binary decision is taken, there is no going back.
- **Penalty assignment accuracy**. We propose to assign penalties to all user activities in the online system. The penalty of an activity should be proportional to the belief that the activity is fraudulent. Fraudsters may predict and exploit errors in the computation of penalties, while errors may also alienate honest users.
- **Malicious behaviors**. Penalties need to be a function also of the computational capabilities of the devices from which the online system activities are performed. Fraudsters may obfuscate the computational capabilities of their devices, while even honest users may use multiple devices to perform activities.

1.5 Outline of The Thesis

The remainder of the thesis is organized and summarized as follows: Chapter 2 provides an overview of the existing literature in the field of fraud detection and prevention. It presents methods utilizing a variety of different techniques and algorithms and then we describe the key difference between our work and the existing studies.

Chapter 3, 4, 5 and 6 represent the main contribution of the thesis. In chapter 3, we present a longitudinal study of Google Play app metadata can provide unique information that is not available through the standard approach of capturing a single app snapshot. It provides insights into fraudulent app promotion and malware indicator behaviors by extracting features from a longitudinal app analysis (e.g., permission, price, update, download count changes).

In Chapter 4 we tackle the problem of identifying both malware and search rank fraud subjects in Google Play. We uncover these nefarious acts by picking out fraudulent and malicious behaviors leave behind telltale signs on app markets, unlike existing solutions.

Chapter 5 introduces the concept of *fraud preemption* systems, solutions deployed to defend online systems such as social networks and app markets. Earlier in this chapter, we describe the system and adversarial model and potential challenges which we need to address in the following sections. Following sections depict important modules of our proposed fraud preemption system which leverage bitcoin inspired computational puzzles to discourage fraud.

Chapter 6 introduces the *fraud de-anonymization* problem, a new approach to address the limitations of status quo solutions, through disincentivizing search rank fraud workers and their employers. We seeks to attribute detected search rank fraud to the humans who posted it and thus provide counter-incentives both for crowdsourcing workers to participate in fraud jobs, and for product developers to recruit fraudsters.

In Chapter 7 we survey the assumptions made by the fraud detection research community on the capabilities and behaviors of fraudsters and study their relevance to app search optimization fraud conducted for Google Play apps. For this, we designed and conducted in-depth interviews with expert fraudsters recruited from several freelancing sites. In addition, we have validated these interviews results with direct empirical data collected from the social networking site, establishing ground

truth for future research. Furthermore, we have identified novel assumptions that seem critical to the detection and de-anonymization of the fraudulent communities studied. We have similarly analyzed and validated these new assumptions which provide novel insight into the techniques and methods used by fraudsters in the wild.

CHAPTER 2

Literature Survey

In this chapter, we survey related studies on search rank fraud. First, we survey existing solutions and literature that analyze the data and show that online marketplace content dynamics. Then, existing solutions to detect malware and fraud products in the online e-commerce site. Their limitations in capturing real-time search rank fraud activities are analyzed. Secondly, in this chapter, we survey the assumptions made in the literature about the fraudulent behaviors perpetrated in online systems, in particular, related to search rank fraud. Thirdly, we study previous solutions on computation based fraud preemption and also graph based fraud detection techniques which have been used extensively to model relationships and detect fraudulent behaviors in online systems. Finally, we study fraud attribution problem and different fraud detection solutions and their re-active behavior and limitations.

2.1 App Market Temporal Dynamics

Viennot et al. [VGN14] developed PlayDrone, a crawler to collect Google Play data. Their main finding is that Google Play developers often include secret key information in the released apps, making them vulnerable to attacks. They further analyze the data and show that Google Play content evolves quickly in time, that 25% of apps are clones, and that native experience correlates strongly to popularity. The analysis is performed over data collected for 3 non-contiguous months (May/June 2013 and November 2013). In contrast, our analysis is performed over apps monitored daily over more than 6 months. Furthermore, our analysis includes orthogonal app market dynamics dimensions, that include the frequency and cycles of app updates, the developer impact and control on the app market, and the dynamics of top-k lists.

Zhong and Michahelles [ZM13] analyze a dataset of Google Play transactions, and suggest that Google Play is more of a "Superstar" market (i.e., dominated by pop-

ular hit products) than a "Long-tail" market (i.e., where unpopular niche products contribute to a substantial portion of popularity). In addition, Zhong and Michahelles [ZM13] show that certain expensive professional apps attract disproportionately large sales. This is consistent with our finding that a few developers are responsible for the most popular apps.

Möller et al. [MMD+12] use an app they posted on Google Play to study the correlation between published updates and their actual installations. They show that 7 days after a security update is published, almost half of the app's users still use an older, vulnerable version. Liu et al. [LAC12] use a dataset of 1,597 ranked mobile apps to conclude that the "freemium" strategy is positively associated with increased sales volume and revenue of the paid apps. Moreover, they show that free apps that rate higher contribute to higher revenue for the paid version. We note that our work studies a multitude of previously unanswered questions about Google Play, regarding app update frequency and pricing appropriateness, and the evolution of top-k lists.

Petsas et al. [PPP+13] explored mobile app markets in the context of 4 providers, that do not include Google Play. They show that the distribution of app popularity deviates from Zipf, due in part to a strong temporal affinity of user downloads to app categories. They show that on the markets they studied, paid apps follow a different popularity distribution than free apps. In contrast, our work exclusively analyzes Google Play, the most popular Android app market. In addition, we focus on different dimensions: (i) app update frequency and its effect on app pricing and resource consumption, (ii) the control of the market and the effect of developer actions on the popularity of their apps and (iii) the evolution in time of top apps and top-k app lists.

Xu et al. [XEG+11] use IP-level traces from a tier-1 cellular network provider to understand the behavior of mobile apps. They provide an orthogonal analysis of spatial and temporal locality, geographic coverage, and daily usage patterns.

Security has been a theme in the large scale collection of mobile apps. Previous work includes malware detection [ZWZJ12], malware analysis [ZJ12], malicious ad libraries [GZJS12], vulnerability assessment [EOMC11], overprivilege identication [FCH+11] and detection of privacy leaks [EGC+10]. While in this paper we focus on the different problem of understanding the dynamics of Google Play, we also introduce novel mobile app attacks.

2.2 Android Malware and Fraud Apps Detection

Zhou and Jiang [ZJ12] collected and characterized 1,200 Android malware samples, and reported the ability of malware to quickly evolve and bypass the detection mechanisms of anti-virus tools.

Burguera et al. [BZNT11] used crowdsourcing to collect system call traces from real users, then used a "partitional" clustering algorithm to classify benign and malicious apps. Shabtai et al. [SKE+12] extracted features from monitored apps (e.g., CPU consumption, packets sent, running processes) and used machine learning to identify malicious apps. Grace et al. [GZZ+12] used static analysis to efficiently identify high and medium risk apps.

Previous work has also used app permissions to pinpoint malware [SLG+12, PGS+12, YSM14]. Sarma et al. [SLG+12] use risk signals extracted from app permissions, e.g., rare critical permissions (RCP) and rare pairs of critical permissions (RPCP), to train SVM and inform users of the risks vs. benefits tradeoffs of apps. FairPlay significantly improves on the performance achieved by Sarma et al. [SLG+12].

Peng et al. [PGS+12] propose a score to measure the risk of apps, based on probabilistic generative models such as Naive Bayes. Yerima et al. [YSM14] also use features extracted from app permissions, API calls and commands extracted from the app executables.

Sahs and Khan [SK12] used features extracted from app permissions and control flow graphs to train an SVM classifier on 2000 benign and less than 100 malicious apps. Sanz et al. [SSL+13] rely strictly on permissions as sources of features for several machine learning tools. They use a dataset of around 300 legitimate and 300 malware apps.

Google has deployed Bouncer, a framework that monitors published apps to detect and remove malware. Oberheide and Miller [OM12] have analyzed and revealed details of Bouncer (e.g., based in QEMU, using both static and dynamic analysis). Bouncer is not sufficient - our results show that 948 apps out of 7,756 apps that we downloaded from Google Play are detected as suspicious by at least 1 anti-virus tool. In addition, FairPlay detected suspicious behavior for apps that were not removed by Bouncer during a more than 6 months long interval.

Instead of analyzing app executables, FairPlay employs a relational, linguistic and behavioral approach based on longitudinal app data. FairPlay's use of app permissions differs from existing work through its focus on the temporal dimension, e.g., changes in the number of requested permissions, in particular the "dangerous" ones. We observe that FairPlay identifies and exploits a new relationship between malware and search rank fraud.

Graph based approaches have been proposed to tackle opinion spam [YA15a, ACF13a]. Ye and Akoglu [YA15a] quantify the chance of a product to be a spam campaign target, then cluster spammers on a 2-hop subgraph induced by the products with the highest chance values. Akoglu et al. [ACF13a] frame fraud detection as a signed network classification problem and classify users and products, that form a bipartite network, using a propagation-based algorithm.

FairPlay's relational approach differs as it identifies apps reviewed in a contiguous time interval, by groups of users with a history of reviewing apps in common. FairPlay combines the results of this approach with behavioral and linguistic clues, extracted

from longitudinal app data, to detect both search rank fraud and malware apps. We emphasize that search rank fraud goes beyond opinion spam, as it implies fabricating not only reviews, but also user app install events and ratings.

2.3 Computation Based Fraud Preemption

Dwork and Naor [DN92] were the first to propose the use of computation to prevent fraud, in particular spam, where the sender of an e-mail needs to include the solution to a "moderately hard function" computed over a function of the e-mail. Juels and Brainard [JB99] proposed to use puzzles to prevent denial of service attacks, while Borisov [Bor06] introduced puzzles that deter Sybils in peer-to-peer networks. In Borisov [Bor06], newly joined peers need to solve a puzzle to which all the other peers have contributed.

FraudSys not only seeks to adapt computational puzzles to prevent online system fraud, but also needs to solve the additional challenges of building puzzles whose difficulty is a function of the probability that an activity is fraudulent, while handling heterogeneous user devices (e.g., ranging from smartphones to machines that specialize in such puzzles).

FraudSys also needs to minimally impact the experience of honest users.

Similarities to recaptcha. FraudSys is similar to Turing test to deter spam users in recaptcha. In addition, legit users who pass a turing test can register a new account in recaptcha while legit users who solves a bitcoin piece can post SN activities such as reviews or Likes. Also, a recognized OCR by legit users in recaptcha rewards OCR researchers, while a solved bitcoin block rewards the service providers.

Graphs have been used extensively to model relationships and detect fraudulent behaviors in online systems. Ye and Akoglu [YA15a] quantified the chance of a subject to be a spam campaign target, then clustered spammers on a 2-hop subgraph

induced by the subjects with the highest chance values. Lu et al. [LZXL13] proposed a belief propagation approach implemented on a review-to-reviewer graph, that simultaneously detects fake reviews and spammers (fraudsters).

Mukherjee et al. [MLG12] proposed a suite of features to identify reviewer groups, as users who review many subjects in common but not much else, post their reviews within small time windows, and are among the first to review the subject. Hooi et al. [HSB+16b] have recently shown that fraudsters have evolved to hide their traces, by adding spurious reviews to popular items. To identify "camouflaged" fraud, Hooi et al. [HSB+16b] introduced "suspiciousness" metrics that apply to bipartite user-to-item graphs, and developed a greedy algorithm to find the subgraph with the highest suspiciousness. Akoglu et al. [ATK15] survey graph based online fraud detection. [FH16] provide a survey of community detection methods, evaluation scores and techniques for general networks.

Zafarani and Liu [ZL15] attempt to detect malicious users using minimal information (i.e., mostly the user's name), in an effort to devise a solution that is applicable to most social networks. This makes it suitable to detect malicious users at registration, as users need to provide at least their username. Instead of malicious users, FraudSys seeks to detect fraudulent actions, then penalizes them according to their likelihood of being fraudulent.

Unlike previous work, FraudSys assigns fraud scores to individual user activities in *real time*, thus uses only partial information. To achieve this, FraudSys develops and leverages features that quantify the connectivity of the user activity to other groups of activities *previously* performed by other fraudsters on the same subject. Further, FraudSys also imposes computation and temporal penalties to discourage fraud creation.

2.4 Online Social Fraud Detection

In 2011, Facebook revealed its use of an "immune system" (FIS) that employs adversarial learning to combat fraud: lengthen the phases controlled by the defender and shorten the phases controlled by the attackers [SCM11]. Google Play is likely to use a similar immune system. For instance, unlike Yelp, Google Play does not reveal if and which user activities are filtered, thus making it difficult to reverse engineer their fraud detection features. However, by identifying 2,664 active, fraudster controlled accounts, DOLOS reveals significant fraud detection error rates in Google Play. DOLOS introduces a departure from existing solutions, as it identifies a ground truth seed of fraud through interaction with active crowdsourcing fraudsters. In the following we describe related work on community detection, and graph and linguistics based fraud detection.

2.4.1 Community Based Fraud Detection

Yang et al. [YHZ+12] analyzed 2,000 "criminal" Twitter accounts and shown that they tend to form small-world social network. Mukherjee et al. [MLG12, MKL+13a] confirmed this finding and introduced features that identify reviewer groups, who review many products in common but not much else, post their reviews within small time windows, and are among the first to review the product.

Beutel et al. [BXG+13] proposed CopyCatch, a system that identifies *lockstep behaviors*, i.e., groups of user accounts that act in a quasi-synchronized manner, to detect fake page likes in Facebook. Cao et al. [CYYP14] also exploited lockstep behaviors and insider information, to cluster user accounts based on similarity actions, and uncover groups of malicious accounts in Facebook. Li et al. [LMC+16] extend lockstep behavior identification with semi-supervised learning based on local spectral graph diffusion, to detect YouTube fraud. In [MKL+13a], they further leverage model

inference and the different behavioral distributions of review spammers vs. non-spammers, to learn the population distributions of the two clusters.

Yang et al. [YHZ+12] also proposed a criminal account inference algorithm that detects new criminal accounts by propagating malicious scores from a seed set of known criminal accounts to their followers according to the closeness of social relationships and the strength of semantic coordination. DOLOS uses a supervised learning based guilt-by-association process to expand the seed fraudster-controlled account sets, which enables us to evaluate its accuracy. It further differs in its combination with a fraud attribution process: assign the newly discovered accounts to the fraudster who controls them.

In LinkedIn, Xiao et al. [XFH15] use features extracted from user-generated text (e.g., name, email, company or university) to train a supervised machine learning pipeline and classify an entire cluster of accounts as malicious or legitimate. Rayana and Akoglu [RA15a, RA16] propose a hybrid, semi-supervised framework that uses graph data (neighborhood density, proximity to other uncertain nodes), relational data (user-review-product graph), behavioral and text data, to spot opinion spam.

To differentiate spammers from legitimate users in social network Bhat et al. [BA13] use community-based structural features by exploiting social network characteristics of community formation. Fortunato and Hric [FH16] provide a survey of community detection methods, evaluation scores and techniques for general networks.

In contrast to previous work, we show that search rank fraud (in Google Play) is perpetrated by crowdsourcing site workers who often control hundreds of user accounts. Then, Dolos goes beyond the simple detection of fraudulent account communities, and instead seeks to de-anonymize the prolific crowdsourcing site workers responsible for signi cant fraud in Google Play.

2.4.2 Graph Based Fraud Detection

Graphs have been used extensively to model relationships and detect fraudulent behaviors e.g., [YA15a, SHY+17]. Ye and Akoglu [YA15a] quantified the chance of a product to be a spam campaign target, then clustered spammers on a 2-hop subgraph induced by the products with the highest chance values. Wang et. al [WGF17] leaveraged a novel Markov Random Field to detect fraudsters in social networks via guilt-by-association on directed graphs. Shen et al [SHY+17] introduced "k-triangles" to measure the tenuity of account groups and proposed algorithms to approximate the Minimum k-Triangle Disconnected Group problem.

Wang et al. [WXLY11] introduced used "heterogeneous review graphs" that capture relations among reviewers, reviews and subjects, and develop an iterative model to identify suspicious reviewers. Malliaros et al. [MMF12] exploit *expansion properties* of large social graphs to build an efficient algorithm for computing the robustness property of time evolving graph to detect communities and anomalies. Faloutsos et al. [Fal14] use static and temporal properties (e.g. eigenspokes) of time-evolving graphs to spot suspicious activities and show the way to handle time-evolving graphs as tensors and large tensors in map-reduce environments.

Hooi et al. [HSB+16b] have shown that fraudsters have evolved to hide their traces, by adding spurious reviews to popular items. They introduced a class of "suspiciousness" metrics that apply to bipartite user-to-item graphs, and developed a greedy algorithm to find the subgraph with the highest suspiciousness metric.

Dolos builds on the empirical observation that search rank fraud is perpetrated by many fraudsters who often control hundreds of user accounts. Dolos seeks to not only detect fraud but also to de-anonymize the influential fraudsters. Instead of heterogeneous or bipartite graphs, Dolos uses co-activity graphs whose nodes correspond to user accounts and weighted edges capture co-activity activities of their end-

point nodes. Thus, techniques such as dense subgraph detection [Cha00, Tso15] and belief propagation [ACF13a] can be used to identify fraudster-controlled accounts. We adapt (1) a dense subgraph detection (DSG) algorithm proposed by Tsourakis [Tso15] to the fraud-component detection problem and (2) a belief propagation algorithm (LBP) to the problem of detecting fraudster-controlled accounts. We show that Dolos' MCDense algorithm significantly outperforms DSG, discovering at least 90% of the accounts controlled by at least 90% of their workers, for double the apps than DSG. While LBP can accurately identify fraudulent accounts, it cannot attribute them to their owners.

2.4.3 Linguistic Based Fraud Detection

Dolos fuses elements from computational linguistics, e.g., [OCCH11, LLK+11], and author de-anonymization, e.g., [OG16]. Ott et al. [OCCH11] used computational linguistics features to detect deceptive TripAdvisor reviews. Lau et al. [LLK+11] proposed a text mining model integrated into a semantic language model to detect fake Amazon reviews. Lappas et al. [LCT12] proposed an incremental greedy method to find a small but diverse set of reviews that together preserve the statistical properties of an entire review corpus. We posit that Dolos, that builds a corpus of fake reviews for each detected fraudster, could benefit from this approach. For instance, Dolos could attribute a new fake account to the fraudster whose "core" of fake reviews is not changed by the addition of the new account's reviews.

Hu et al. [HTGL14] proposed a matrix factorization based model to detect Twitter spammers, that analyzed sentiment differences between spammers and normal users. Sentiment and bias, e.g., [LCN15] may complement stylometry tools to help attribute detected fraud. Overdorf and Greenstadt [OG16] proposed authorship attribution methods that work across social networks. Abbasi and Chen [AC08] de-

veloped Writeprints, a system for de-anonymizing e-mail, IM, reviews and program code. Narayanan et al. [NPG+12] proposed author de-anonymization techniques that handle huge number of classes (100,000 authors).

DOLOS is the first work to de-anonymize search rank fraud by linking it to crowdsourcing site worker accounts, and to break the anonymity barrier between crowdsourcing site workers and Google Play user accounts. DOLOS overcomes limitations that stem from malicious crowdsourcing behaviors, to break the anonymity barrier between crowdsourcing site workers and Google Play user accounts. To achieve this, DOLOS leverages Google Play features, and traits we identified in fraudulent crowdsourcing behaviors, to introduce new techniques to collect ground truth fraud worker profiles, detect clusters of related fraud, and attribute them to the workers who control them.

Fraud data collection. Previous work, e.g., [GHP+12, OCCH11] used crowdsourcing to collect gold standard review datasets, i.e., by asking workers to write reviews for existing TripAdvisor venues. Rubin and Conroy [RC12] argue that one challenge in building a quality fraud dataset is to incorporate motivation in the outsourced tasks. We address this challenge by collecting existing fraudulent reviews, that were written by expert fraudsters as part of real ASO jobs.

2.5 Crowdsourced Review Fraud Survey

To the best of our knowledge, this is the first work to systematically organize assumptions about fraud behaviors in online systems and validate them using recruited fraud workers and ground truth data.

Previous research work illustrates several characteristics of incentivized reviews which helps us to generate the assumptions and the questionnaire afterward. The most important and commonly seen behavior is review burst where Groups of users

acting together and reviews are posted within a very short burst [FML+13a, HTS16, LFW+17a, HSB+15, BSLL+16a].

To build the reputation of the spamming accounts fraudsters often write genuine-looking reviews to camouflage/mask their misactivities for adding camouflage edges to random honest users [ACF13a, HSB+16b, RA15b]. Spammers are likely to copy reviews across similar products and use a certain vocabulary to generate fake reviews [MKL+13a, FML+13a]. The freshness of accounts and short-lived accounts is another important characteristic usually seen for opinion spamming [MKL+13a, YKA16]. Also to control the sentiment Spammers try to be the first reviewers for products [MKL+13a, Xu13]. Spammers would change IP addresses frequently when posting reviews to fool the fake review filtering system [LCM+15]. Also, fraudsters develop automated programs or bot to control and create new spam accounts and post reviews from those accounts [SMJ+15].

CHAPTER 3

Longitudinal Study of Google Play

3.1 Motivation

The revolution in mobile device technology and the emergence of "app markets", have empowered regular users to evolve from technology consumers to enablers of novel mobile experiences. App markets such as Google Play provide new mechanisms for software distribution, collecting software written by developers and making it available to smartphone users. This centralized approach to software distribution contrasts the desktop paradigm, where users obtain their software directly from developers.

Developers and users play key roles in determining the impact that market interactions have on future technology. However, the lack of a clear understanding of the inner workings and dynamics of popular app markets, impacts both developers and users. For instance, app markets provide no information on the impact that developer actions will likely have on the success of their apps, or guidance to users when choosing apps, e.g., among apps claiming similar functionality.

This situation is exploited however by fraudulent and malicious developers. The success of Google Play and the incentive model it offers to popular apps [1], make it an appealing target for fraudulent and malicious behaviors. Fraudulent developers have been shown to attempt to engineer the search rank of their apps [Sie14], while malicious developers have been shown to use app markets as a launch pad for their malware [Min14, Mlo14, Rob15, Gre14].

Contributions. In this article we seek to shed light on the dynamics of Google Play, the most popular Android app market. We report results from one of the first characteristic studies on Google Play, using real-world time series data. To this end,

[1] Google offers financial incentives for contribution to app development, by making revenue sharing transparent for developers (70-to-30 cut, where developers get 70% of the revenue).

we have developed GPCrawler, a prototype app market crawling system. We have used GPCrawler to collect data from more than 470,000 Google Play apps, and daily monitor more than 160,000 apps, over more than 6 months.

We use this data to study two key aspects of Google Play. First, we seek to understand the dynamics of the market in general, from an application and developer perspective. For this, we evaluate the frequency and characteristics of app updates (e.g., their effects on bandwidth consumption), and use the results to determine if developers price their apps appropriately. We show that only 24% of the 160,000 app that we monitored have received an update within 6 months, and at most 50% of the apps in any category have received an update within a year from our observation period. We conclude that market inactivity has a significant impact on the price distribution. Therefore, while pricing is an important and complex task, relying on statistics computed on the entire population (as opposed to only active apps) may mislead developers, e.g., to undersell their apps (§3.5.1). Also, we show that typical app update cycles are bi-weekly or monthly. More frequently updated apps (under beta-testing or unstable) can impose substantial bandwidth overhead and expose themselves to negative reviews (§3.5.3).

To evaluate the developer impact, we first seek to verify our hypothesis that a few developers control the app market supply. Our analysis reveals however that developers that create many applications are not creating many popular applications. Instead, we discovered that a few elite developers are responsible for applications that dominate the total number of downloads (§3.6). Second, we evaluate the impact of developer actions on the popularity of their apps. We show that few apps frequently change prices, and with every subsequent software update, a developer is more likely to decrease the price. However, changing the price does not show an observable association with the app's download count(§3.6).

Impact of the study. A longitudinal study of Google Play app metadata can

provide unique information that is not available through the standard approach of capturing a single app snapshot. Features extracted from a longitudinal app analysis (e.g., permission, price, update, download count changes) can provide insights into fraudulent app promotion and malware indicator behaviors. For instance, spikes in the number of positive or negative reviews and the number of downloads received by an app can indicate app search optimization campaigns launched by fraudsters recruited through crowdsourcing sites. Frequent, substantial app updates may indicate Denial of Service (DoS) attacks, while permission changes can indicate benign apps turning malicious see § 3.7.1. Features extracted from a longitudinal app monitoring can be used to train supervised learning algorithms to detect such behaviors.

In addition, a detailed longitudinal study of Google Play apps can improve developer and user experiences. For instance, app development tools can help developers optimize the success of their apps. Such tools can integrate predictions of the impact that price, permissions and code changes will have on the app's popularity, as well as insights extracted from user reviews. In addition, visualizations of conclusions, and analytics similar to the ones we perform in this paper, can help users choose among apps with similar claimed functionality.

We include a detailed discussion of the applicability and future research directions in app market analytics in §3.7.

3.2 Google Play Overview

App Distribution Channel: Google Play is the app distribution channel hosted by Google. Each app submitted by a developer gets an entry on the market in the form of a webpage, accessible to users through either the Google Play homepage or the search interface. This webpage contains meta-information that keeps track of information pertaining to the application (e.g., name, category, version, size, prices).

In addition, Google Play lists apps according to several categories, ranging from "Arcarde & Action" to "Weather". Users download and install apps of interest, which they can then review. A review has a rating ranging from 1 to 5. Each app has an *aggregate rating*, an average over all the user ratings received. The app's webpage also includes its usage statistics (e.g., rating, number of installs, user reviews). This information is used by users when they are deciding to install a new application.

App Development: In order to submit apps to Google Play, an Android developer first needs to obtain a publisher account for a one-time fee of $25. The fee encourages higher quality products and reduces spam [Goo12]. Google does not limit the number of apps that can be submitted by developers. As a measure to reduce spam, Google recently started the Bouncer [gooa] service, which provides automated scanning of applications on Google Play for potential malware. Developers can sell their apps for a price of their choice, or distribute them for free.

Permission Model: Android follows the Capability-based [Lev84] security model. Each app must declare the list of capabilities (permissions) it requires in a manifest file called *Android-Manifest.xml*. When a user downloads an app through the Google Play website, the user is shown a screen that displays the permissions requested by the application. Installing the application means granting the application all the requested permissions i.e. an *all-or-none* approach.

3.3 Data Collection

We use *snapshot* to refer to the entire state of the market i.e., it contains meta information of all apps. We first describe GPCrawler, our app market crawler, then describe the datasets that we collected from Google Play.

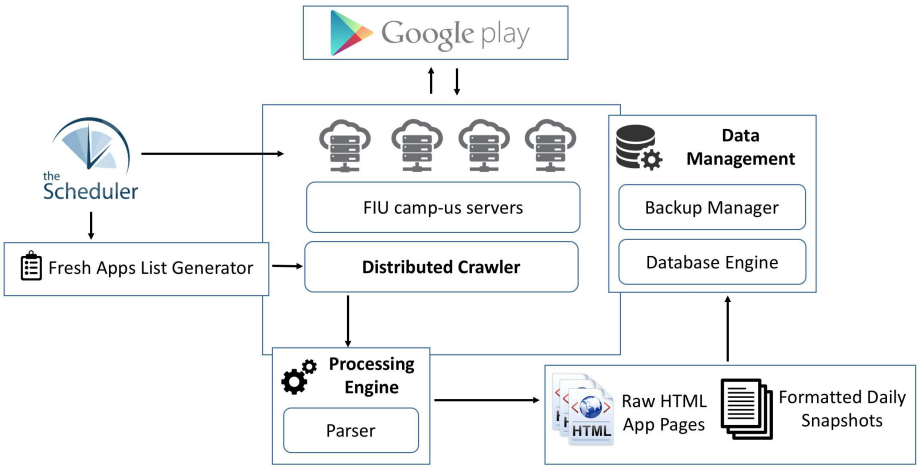
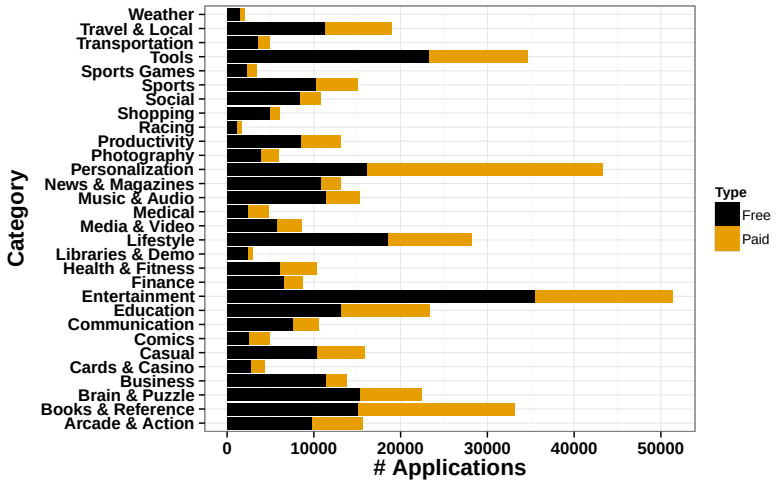


Figure 3.1: Architecture of GPCrawler, the developed GooglePlay crawler. It consists of a distributed crawler, processing engine and data management components.

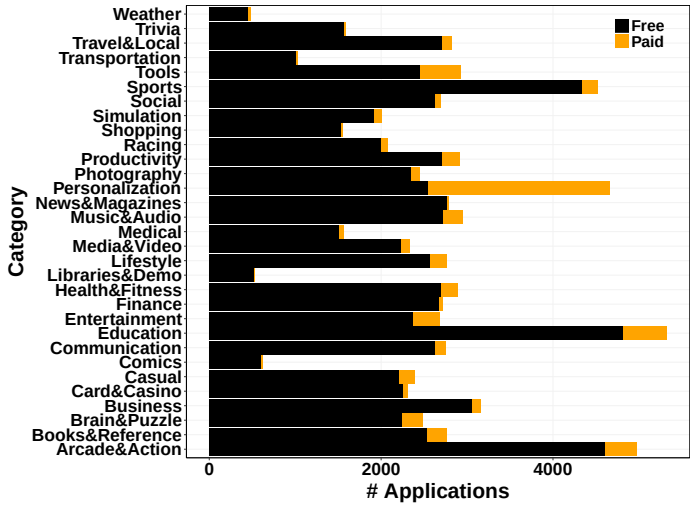
3.3.1 The GPCrawler

GPCrawler, our prototype market crawling system (see Figure 3.1 for an overview) consists of three main components. First, the *Distributed Crawler* component, which is responsible for crawling the target market and collecting information on various apps that are accessible from the current geographical location. We initially leveraged hundreds of foreign proxies to address challenge 3 above. However, we later decided to rely only on local US-based proxies for stability reasons. While this trades-off completeness for consistency, having continuous information about a few apps improves the accuracy of most statistical inference tasks compared to having discrete information about hundreds of thousands of apps.

To seed our distributed crawler, we initially ran it using a list consisting of about 200 randomly hand-picked apps from different categories. To address Challenge 1, our app discovery process is designed as follows: After retrieving each page, the "Similar Apps" portion of the raw HTML page is parsed to obtain a new list of packages. These packages are queued for crawling and simultaneously appended to the previous day's package list. We have also detected a ban detection engine in place that deactivates



(a)



(b)

Figure 3.2: Distribution of free vs. paid apps, by category, for (a) dataset.2012 and dataset.14-15. The number of free apps exceeds the number of paid ones especially in dataset.14-15. We conjecture that this occurs due to user tendency to install more free apps than paid apps. Since 2012, developers may have switched from a direct payment model for paid apps, to an ad based revenue model for free apps.

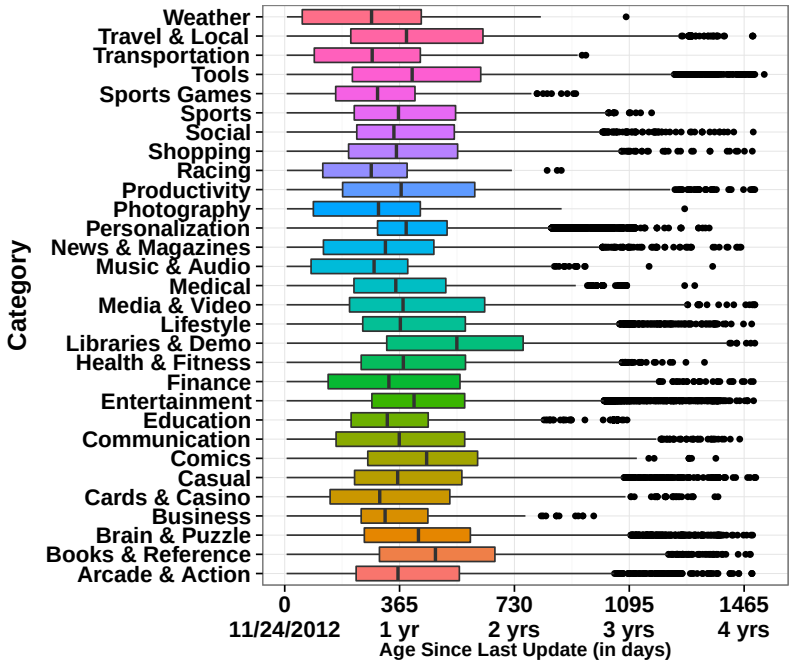
servers once it observes a threshold number of "404 Not Found" messages (Challenge 2) from the market provider.

The second component, the "Processing Engine" contains a *Map-Reduce Parser* component that uses the map-reduce paradigm [DG08] to handle parsing of hundreds of thousands of raw HTML app pages. In the "map" stage, a chunk of files (≈10K) are mapped onto each of the 700 machines and a parser (written in Python) parses these HTML and extracts the meta information. In the "reduce" stage, these individual files are combined into a single file and de-duplicated to maintain data integrity. This stage takes ≈1-1.5 hours. After constructing the aggregate file, we address Challenge 3 using the assertion checker that takes a *best-effort* approach to ensure that all the information has been correctly parsed from the raw files. Note that despite our best-effort approach, our dataset still contained some missing information due to temporary unavailability/maintenance of servers.

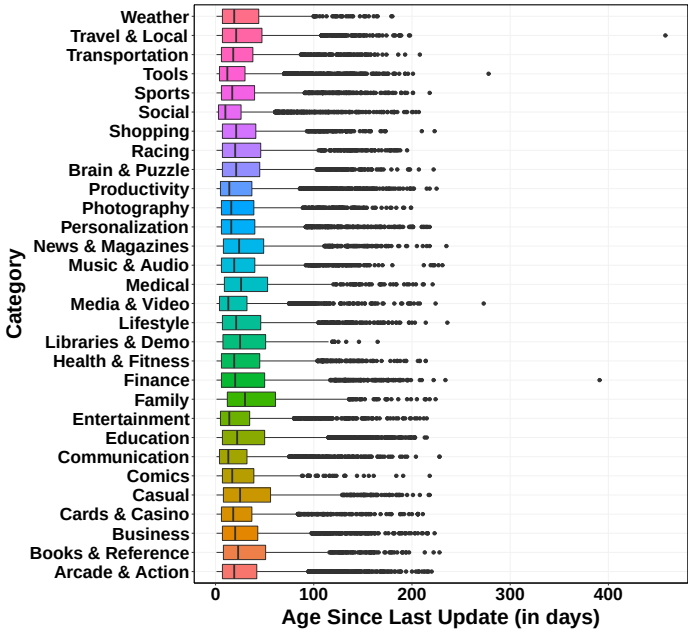
The third, "Data Management" component, archives the raw HTML pages (≈14 GB compressed/day) in a cloud storage to support any ad hoc processing for other tasks (e.g., analyzing HTML source code complexity) and subsequently removed from the main servers. To address Challenge 4, the formatted daily snapshot (≈200 MB/day) is then inserted into a database to support data analytics. We setup the relevant SQL Jobs to ensure that indexes are re-built every two days — this step significantly speeds up SQL queries. Our six months of archived raw files consume ≈7 TB of storage and the database consumes ≈400 GB including index files.

3.4 Data

We used GPCrawler to collect two Google Play datasets, which we call dataset.2012 and dataset.14-15.



(a)



(b)

Figure 3.3: Box and whiskers plot of the time distribution from the last update, by app category, for (a) dataset.2012: at most 50% of the apps in each category have received an update within a year and (b) dataset.14-15: at most 50% of the apps in each category have received an update within 35 days. This may occur since new apps are likely to have more bugs and receive more attention from developers.

3.4.1 Dataset.2012

We have used a total of 700 machines [2] for a period of 7.5 months (February 2012 - November 2012) to collect data from 470,000 apps. The first 1.5 months are the "warm up" interval. We do not consider data collected during this period for subsequent analysis. Instead, we focus on a subset of 160K apps for which we have collected the following data:

`GOOGPLAY-FULL`: We used *GPCrawler* to take daily snapshots of Google Play store from April - November, 2012. For each app, we have daily snapshots of application meta-information consisting of the developer name, category, downloads (as a range i.e., 10-100, 1K-5K etc.), ratings (on a 0-5 scale), ratings count (absolute number of user ratings), last updated timestamp, software version, OS supported, file size, price, url and the set of permissions that the app requests. Figure 3.2(a) shows the distribution of apps by category. While overall, the number of free apps exceed the number of paid apps, several popular categories such as "Personalization" and "Books & References" are dominated by paid apps.

`GOOGPLAY-TOPK`: Google publishes several lists, e.g., *Free* (most popular apps), *Paid* (most popular paid), *New (Free)* (newly released free apps), *New (Paid)* (newly released paid) and *Gross* (highly grossing apps). Each list is divided into ≈20 pages, each page consisting of 24 apps. These lists are typically updated based on application arrival and the schedule of Google's ranking algorithms. Since we cannot be notified when the list changes, we took hourly snapshots of the lists. Our `GOOGPLAY-TOPK` consists of hourly snapshots of five top-k lists (≈ 3000 apps) from Jul-Nov, 2012 (≈2880 hours worth of data).

[2] We have used 700 machines, each with a different IP address and from a different subnet, in order to avoid getting banned during the crawling process.

3.4.2 Dataset.14-15

Further, we have used a dataset of more than 87,000 newly released apps that we have monitored over more than 6 months [RRCC16b]. Specifically, we have collected newly released apps once a week, from Google Play's "New Release" links, to both free and paid apps. We have validated each app based on the date of the app's first review: we have discarded apps whose first review was more than 40 days ago. We have collected 87,223 new releases between July and October 2014, all having less than 100 reviews.

We have then monitored and collected data from these 87,223 apps between October 24, 2014 and May 5, 2015. Specifically, for each app we captured "snapshots" of its Google Play metadata, twice a week. An app snapshot consists of values for all its time varying variables, e.g., the reviews, the rating and install counts, and the set of requested permissions. For each of the $2,850,705$ reviews we have collected from the $87,223$ apps, we recorded the reviewer's name and id, date of review, review title, text, and rating.

Figure 3.2(b) shows the distribution of apps by category. With the exception of the "Personalization" category, the number of free apps significantly exceeds the number of paid apps. We have observed that consistently through our collection effort, we identified fewer top paid than free new releases. One reason for this may be that users tend to install more free apps than paid apps. Thus, not only developers may develop fewer paid apps, but paid apps may find it hard to compete against free versions. We note that free apps bring revenue through ads.

3.5 Popularity and Staleness

We first evaluate the fraction of apps that are active, and discuss the implications this can have on app pricing. We then classify apps based on their popularity, and

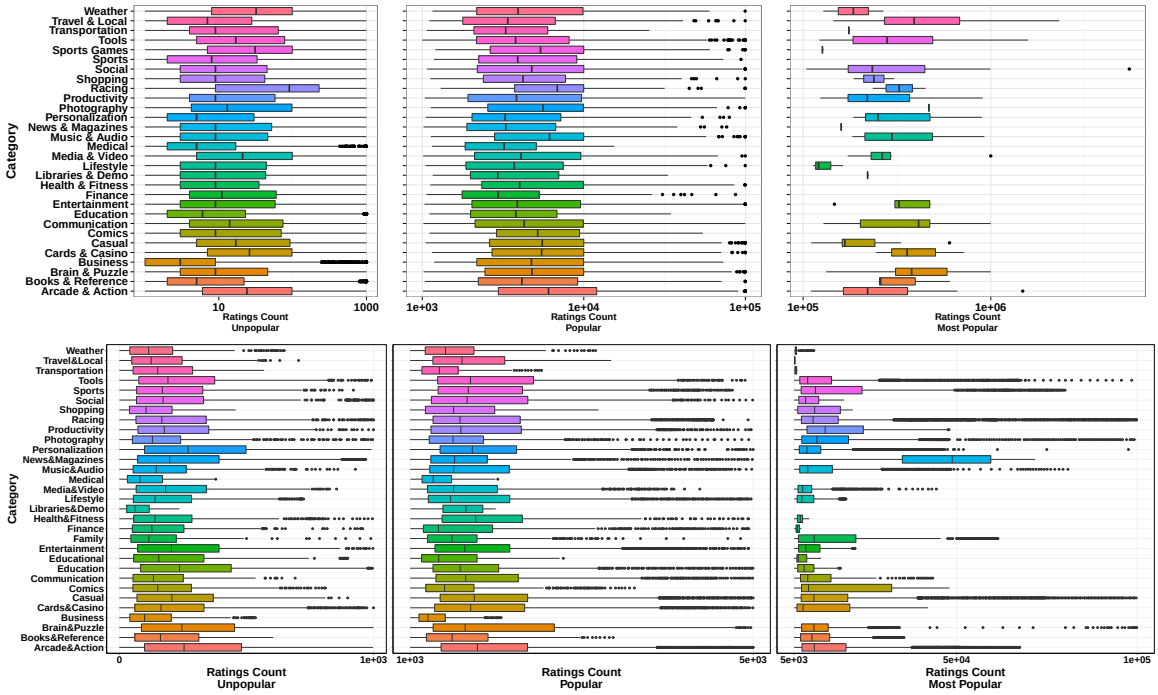


Figure 3.4: Per app category distribution of rating counts. (Top) Dataset.2012. The distribution is almost symmetric in the case of unpopular apps. The distributions for most of the categories are symmetric in the popular class and span roughly from $1,000$ to $100K$ ratings. The Business and Comics categories do not have any apps in the most-popular class. (Bottom) Dataset.14-15. We observe smaller rating counts compared with the apps in dataset.2012. This is natural, as these are new apps, thus likely to receive fewer ratings. We also note that while a few Business, Libraries & Demo and Medical categories are unpopular and popular, none are most popular.

study the distribution of per-app rating counts. Finally, we study the frequency of app updates for apps from various classes and the implications they can have on end-users. All the analysis presented in this section is performed using `GOOGPLAY-FULL`.

3.5.1 Market Staleness

An important property of a market is its "activity", or how frequently are apps being maintained. We say that an app is *stale* if it has not been updated within the last year from the observation period, and *active* otherwise.

The task of setting the app price is complex. However, relying on statistics computed on the entire population, as opposed to only active apps, may mislead developers. For instance, given that the listing price of apps forms a key component of its valuation and sale, this becomes an important factor for fresh developers trying to enter the market. Specifically, the median price in our dataset is $0.99 when all apps are considered and $1.31 when considering only active apps. This confirm our intuition that developers that set their price based on the former value are likely to sell their apps at lower profits.

Figure 3.3(a) shows the box and whiskers plot [Ben88] of the per-app time since the last update, by app category, for dataset.2012. At most 50% of the apps in each category have received an update within a year from our observation period. For instance, most apps in *Libraries & Demo* have not been updated within the last 1.5 years. Some categories such as *Arcade & Action*, *Casual*, *Entertainment*, *Books & Reference*, *Tools* contain apps that are older than three years.

Figure 3.3(b) plots this data for dataset.14-15. Many freshly uploaded apps were uploaded more recently: 50% apps in each category receive an update within 35 days, while apps in the "Social" and "Tools" categories received updates even within 15 days. This is natural, as new apps may have more bugs and receive more developer attention.

Several reasons may explain the lack of updates received by many of the apps we monitored. First, some apps are either stable or classic (time-insensitive apps, not expected to change) and do not require an update. Other apps, e.g., e-books, wallpapers, libraries, do not require an update. Finally, many of the apps we monitored seemed to have been abandoned.

Class	# download	% Dataset.2012	% Dataset.14-15
Unpopular	$0 - 10^3$	74.14	77.55
Popular	$10^3 - 10^5$	24.1	18.43
Most-Popular	$> 10^5$	0.7	4.00

Table 3.1: Popularity classes of apps, along with their distribution. Dataset.14-15 has a higher percentage of most-popular apps.

3.5.2 App Popularity

We propose to use the *download count* to determine app popularity. Higher rating counts mean higher popularity but not necessarily higher quality (e.g., an app could attract many negative ratings). Including unpopular apps will likely affect statistics such as update frequencies: including unpopular apps will lead to a seemingly counter-intuitive finding, indicating that most apps do not receive any updates. Therefore, we classify apps according to their popularity into three classes, "unpopular", "popular" and "most-popular".

Table 3.1 shows the criteria for the 3 classes and the distribution of the apps in dataset.2012 and dataset.14-15 in these classes. The newly released apps have a higher percentage of unpopular apps, however, surprisingly, they also have a higher percentage of "most-popular" apps. This may be due to the fact that the newly released apps are more recent, coming at a time of higher popularity of mobile app markets, and maturity of search rank fraud markets (see § 3.7.1).

Figure 3.4 (top) depicts the distribution of rating counts of apps from dataset.2012, split by categories. We observe that the *Business* and *Comics* categories do not have any apps in the *Most-Popular* class, likely because of narrow audiences. From our data, we observed that the median price of apps ($1.99) in these categories is significantly higher than the population ($1.31) indicating lower competition. The population in other categories is quite diverse with a number of outliers. For instance, as expected, "Angry Birds" and "Facebook" are most popular among the *Most-Popular*

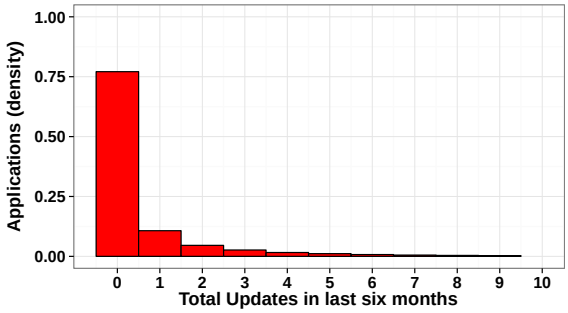
class for *Arcade & Action* and *Social* categories, respectively. On the other hand, the distribution is almost symmetric in case of *Unpopular* except *Business* and *Medical* categories where there are a number of outliers that are significantly different from the rest of the population. We found that these are trending apps — apps that are gaining popularity. For instance, the free app "Lync 2010" from "Microsoft Corporation" in *Business* has 997 ratings. In case of *Popular*, the distributions for most of the categories are symmetric and span roughly from $1,000$ to $100K$ ratings where 75% of apps have less than $10,000$ rating counts except *Arcade & Action* category.

Figure 3.4 (bottom) shows the same distribution for the apps in dataset.14-15. We emphasize that the distribution is plotted over the ratings counts at the end of the observation interval. Since these are newer apps than those in dataset.2012, it is natural that they receive fewer ratings. We also observe that several categories do not have apps that are in the "most popular" category, including the "Business", "Libraries & Demo" and "Medical" categories.

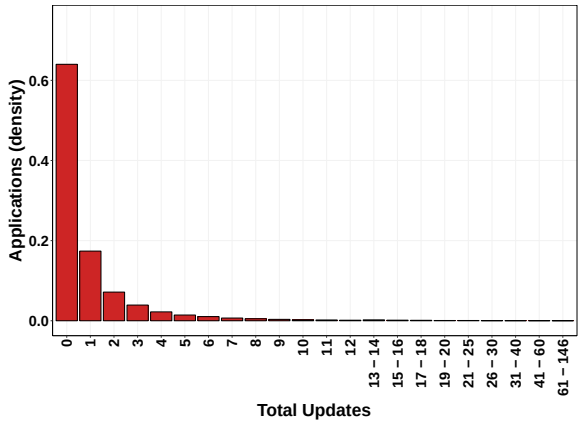
3.5.3 App Updates

Updates form a critical and often the last part of the software lifecycle [GJM02]. We are interested in determining if mobile app developers prefer seamless updating i.e., if they push out releases within short time periods.

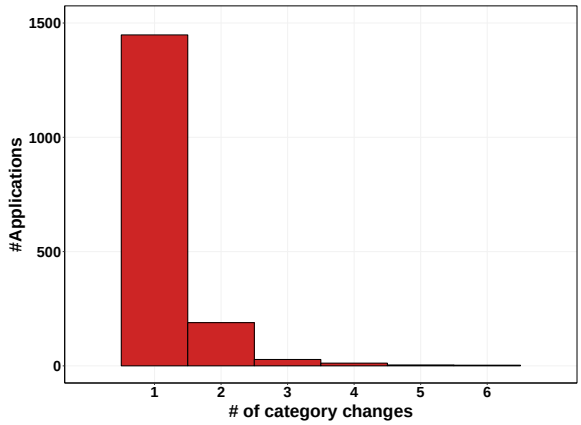
Fig. 3.5(a) shows the distribution of the number of updates received by the apps in dataset.2012. Only 24% apps have received at least one update within our observation period — nearly 76% have never been updated. In contrast, Fig. 3.5(b) shows that 35% of the "fresh" apps in dataset.14-15 have received at least one update within our observation period. Several apps received more than 100 updates, with one app receiving 146 updates in a 6 months interval. We conjecture that this occurs because



(a)



(b)



(c)

Figure 3.5: (a) Histogram of app updates for dataset.2012. Only 24% apps have received at least one update between April-November 2012. (b) Histogram of fresh app updates for dataset.14-15. Unlike the dataset.2012, 35% of the fresh apps have received at least one update, while 1 app received 146 updates! (c) Histogram of app category changes. 1.9% apps have received at least one category change between October 24, 2014 and May 5, 2015, while several have received 6 category changes.

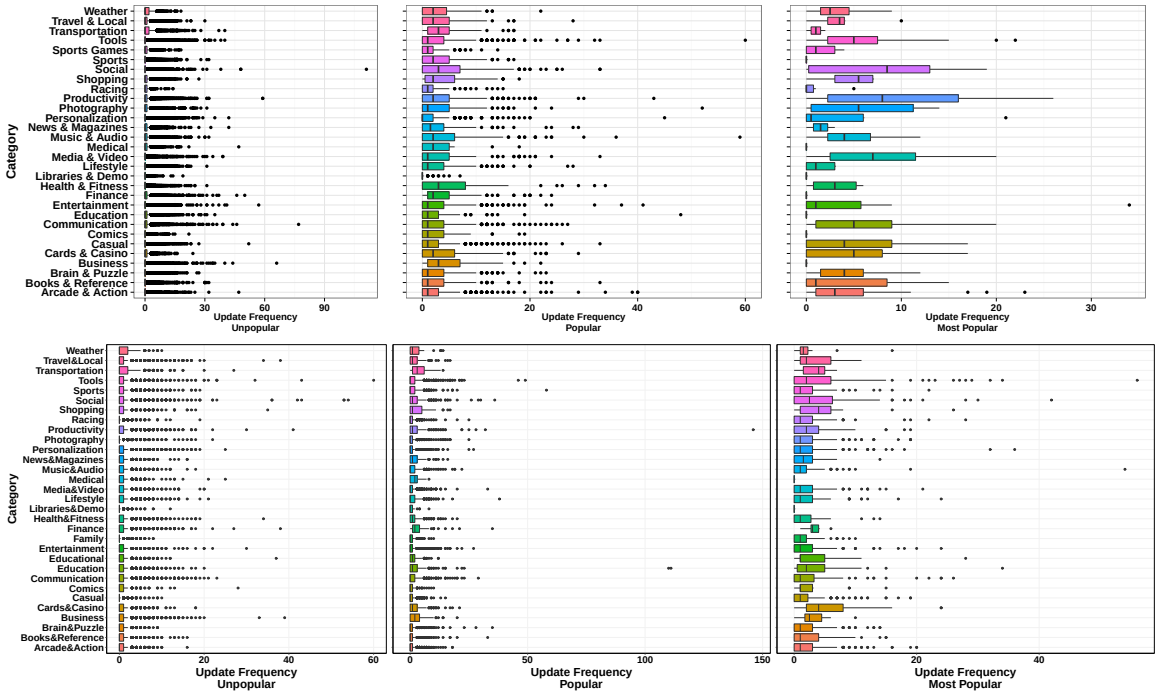


Figure 3.6: The distribution of update frequency, i.e., the update count for each app per category. (Top) Dataset.2012. Unpopular apps receive few or no updates. Popular apps however received more updates than most-popular apps. This may be due to most-popular apps being more stable, created by developers with well established development and testing processes. (Bottom) Dataset.14-15. We observe a similar update count distribution among unpopular apps to dataset.2012. Further, in the popular and most popular classes, most app categories tend to receive fewer updates than the dataset.2012 apps.

these are newly released apps, thus more likely to have bugs, and to receive attention from their developers.

Figure 3.6 (top) plots the distribution of the update frequency of the apps from dataset.2012, across categories based on their popularity. As expected, *Unpopular* apps receive few or no updates. We observed that this is due to the app being new or abandoned by its developer. For instance, "RoboShock" from "DevWilliams" in *Arcade & Action* with good reviews from 4 users has received only one update on September 28, 2012 since its release in August 2011 (inferred from its first comment). Another app "Shanju" from "sunjian" in *Social* has not been updated since May 27, 2012 even though it received negative reviews.

Outliers (e.g., "Ctalk" in the *Social* category) push out large number of updates (111). Popular apps are updated more frequently: 75% in each category receive 10 or fewer updates, while some apps average around 10-60 updates during our observation period. User comments associated with these apps indicate that the developer pushes out an update when the app attracts a negative review (e.g., "not working on my device!"). In the *Most-Popular* category, the population differs significantly. While some apps seldom push any updates, apps like "Facebook" (*Social*) have been updated 17 times. The lower number of updates of most popular apps may be due to testing: Companies that create very popular apps are more likely to enforce strict testing and hence may not need as many updates as other apps.

To identify how frequently developers push these updates, we computed the average update interval (AUI) per app measured in days (figure not shown). In *Popular* and *Unpopular* classes, 50% of apps receive at least one update within 100 days. The most interesting set is a class of Unpopular apps that receive an update in less than a week. For instance, the developer of "Ctalk" pushed, on average, one update per day totaling 111 updates in six months indicating development stage (it had only 50-100 downloads) or instability of the app. On the other hand, *Most-Popular* apps receive an update within 20 to 60 days.

Figure 3.6 (bottom) shows the update frequency for the newly released apps of dataset.14-15. Compared to the apps in dataset.2012, new releases exhibit a similar update frequency distribution, with slightly lower third quartiles. However, a few newly released popular apps receive significantly more updates, some more than 100 updates.

Updates, bandwidth and reputation. A high update frequency is a likely indicator of an on-going beta test of a feature or an unstable application. Such apps have the potential to consume large amounts of bandwidth. For instance, a music player "Player Dreams", with 500K-1M downloads, pushed out 91 updates in the last six

months as part of its beta testing phase (inferred from app description). With the application size being around 1.8 MB, this app has pushed out ≈164 MB to each of its users. Given its download count of 500K-1M, each update utilizes ≈0.87-1.71 TB of bandwidth. We have observed that frequent updates, especially when the app is unstable, may attract negative reviews. For instance, "Terremoti Italia" that pushed out 34 updates in the observation interval, often received negative reviews of updates disrupting the workflow.

Furthermore, app market providers can use these indicators to inform users about seemingly unstable applications and also as part of the decision to garbage collect abandoned apps.

3.5.4 App Category Changes

In the fresh app dataset.14-15 we found app category change events, e.g., "Social" to "Communication", "Photography" to "Entertainment", between different game subcategories. Such category changes may enable developers to better position their apps and improve on their install and download count, as categories may overlap, and apps may stretch over multiple categories. Fig. 3.5(c) shows the distribution of the number of app category changes recorded over the 6 months in dataset.14-15. Only 1.9% of apps have received at least one category change.

3.6 Developer Impact

In this section, we are interested in understanding what fraction of popular apps are being controlled by an elite set of developers and if there is a power-law effect in-place. Next, we analyze the impact that developer actions (e.g., changing the price, permissions etc.) can have on the app popularity. We use dataset.2012 for this analysis.

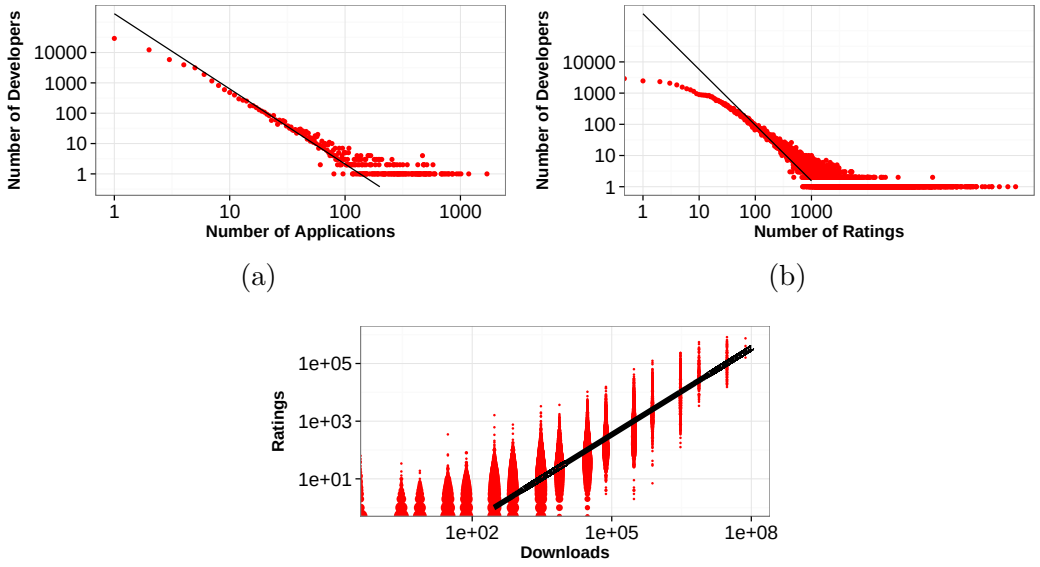


Figure 3.7: (a) Distribution of apps per developer. (b) Distribution of total reviews per developer. (c) Scatter plot of downloads vs. ratings in Google Play. Both axes are log-scaled. A linear curve was fitted with a slope of 0.00341 indicating that an application is rated once for about every 300 downloads.

3.6.1 Market Control

To understand the impact that developers have on the market, we observe their number of apps, downloads, and review count. Figure 3.7 plots these distributions, all showing behavior consistent with a power-law distribution [Mit04]. We display the maximum likelihood fit of a power-law distribution for each scatter plot as well [JW02, CSN07]. Figure 3.7(a) shows that a few developers have a large number of apps while many developers have few apps. However, the developers that post the most apps do not have the most popular apps in terms of reviews and download counts. Instead, Figure 3.7(b) shows that a few developers control apps that attract most of the reviews. Since Figure 3.7(c) shows an almost linear relation between review and download counts (1 review for each 300 downloads), we conclude that the apps developed by the controlling developers are popular.

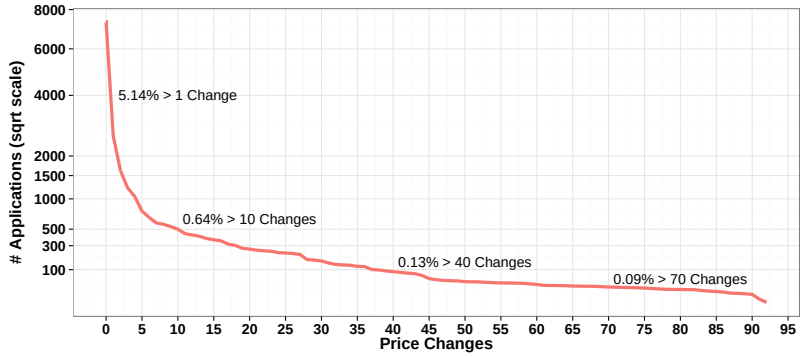


Figure 3.8: The (square root) of the number of apps whose number of price changes exceeds the value on the x axis. Only 5.14% of the apps had a price change, and 0.09% of the apps had more than 70 changes.

3.6.2 Price Dispersion

Menu costs (incurred by sellers when making price changes) are lower in electronic markets as physical markets incur product re-labeling costs [LBDV97]. In app markets menu costs are zero. We now investigate if developers leverage this advantage i.e., if they adjust their prices more finely or frequently.

Figure 3.8 shows a variation of the complementary cumulative distribution frequency (CCDF) of the number of price changes an app developer made during our observation period. Instead of probabilities, the y axis shows the square root of the number of apps with a number of price changes exceeding the value shown on the x axis. We observe that 5.14% of the apps (≈4000) have changed their price at least once. The tail ($>$ 70 changes) is interesting — about 23 apps are frequently changing their prices. From our data, we observed that they are distributed as follows: *Travel & Local* (11), *Sports* (5), *Business* (2), *Brain & Puzzle* (2) and one in each of *Education*, *Finance*, and *Medical*. In this sample, "LogMeIn Ignition", developed by LogMeIn, has 10K-50K downloads and underwent 83 price changes (Min:$18.44, Max:$27.80, Avg:$26.01, Stdev:$2.01). The rest were either recently removed or are unpopular.

Price dispersion is the spread between the highest and lowest prices in the mar-

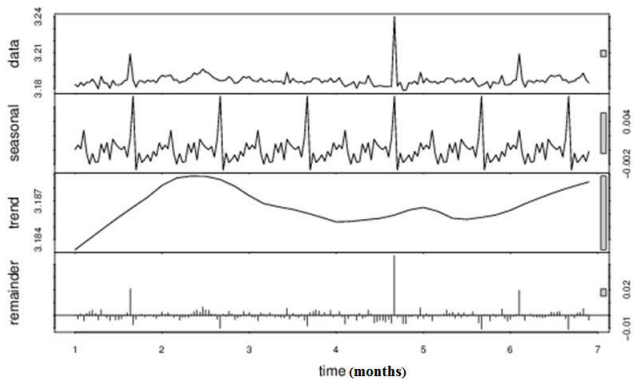


Figure 3.9: Monthly trend for the average app price. Over the 6 month observation interval, the average app price does not exhibit a monthly trend.

ket. In our dataset, we used the *coefficient of variation* (COV) [WSB04], the ratio of standard deviation to the mean, to measure price dispersion. COV= 1 indicates a dispersal consistent with a Poisson process i.e., uniformly at random; COV> 1 indicates greater variability than would be expected with a Poisson process; and COV< 1 indicates less variation. In our dataset, we observed an average COV (computed for all apps) to be 2.45 indicating a non-negligible price dispersion, in agreement with results in the context of other electronic markets [BS00].

Figure 3.9 shows the STL decomposition [CCMT90] of the average price time series in the observation interval, for a periodicity of one month. The gray-bar on the "monthly panel" (see Figure 3.9) is only slightly larger than that on the "data" panel indicating that the monthly signal is large relative to the variation in the data. In the "trend" panel, the gray box is much larger than either of the ones on the "data"/"monthly" panels, indicating the variation attributed to the trend is much smaller than the monthly component and consequently only a small part of the variation in the data series. The variation attributed to the trend is considerably smaller than the stochastic component (the remainders). We deduce that in our six month observation period this data does not exhibit a trend.

3.6.3 Impact of Developer Actions

Developers have control over several attributes they can leverage to increase the popularity of their apps, e.g., pricing, the number of permissions requested from users and the frequency of updates. In this section we investigate the relation between such levers and their impact on app popularity. For instance, common-sense dictates that a price reduction should increase the number of downloads an app receives.

We study the association between app attribute changes. We define a random variable for increase or decrease of each attribute, and measure the association among pairs of variables. For example, let X be a variable for price increase. For each $\langle$ day, app $\rangle$ tuple, we let X be a set of all of the app and day tuples where the app increased its price that day (relative to the previous day's value). For this analysis we consider 160K apps that have changed throughout our observation period, and we discard the remaining apps. We use the Yule measure of association[War08] to quantify the association between two attributes, A and B: $\frac{|A\cap B|*|\overline{A}\cap\overline{B}|-|A\cap\overline{B}|*|\overline{A}\cap B|}{|A\cap B|*|\overline{A}\cap\overline{B}|+|A\cap\overline{B}|*|\overline{A}\cap B|}$.

$\overline{A}$ is the complement of A, i.e., each $\langle$ day, app $\rangle$ tuple where the attribute does not occur, and $|A|$ denote the cardinality of a set (in this case A). This association measure captures the association between the two attributes: zero indicates independence, +1 indicates perfectly positive association, and -1 perfectly negative association.

We observed that changing the price does not show significant association with the download or review counts. We randomly sampled 50 apps where this is happening and observe the following to be the main reasons. First, apps are initially promoted as free and a paid version is released if they ever become popular. However, in some cases, the feature additions are not significant (e.g., ads vs. no ads) and hence do not cause enough motivation for users to switch to the paid version. Second, with app markets offering paid apps for free as part of special offers (e.g., Thanksgiving deals), users may expect the app to be given out for free rather than take a discount of a

few cents.

3.7 Research Implications

We now discuss the implications of longitudinal monitoring on security and systems research in Android app markets.

3.7.1 Fraud and Malware Detection

App markets play an essential role in the profitability of apps. Apps ranked higher in the app market become more popular, thus make more money, either through direct payments for paid apps, or through ads for free apps. This pressure to succeed leads some app developers to tinker with app market statistics known to influence the app ranking, e.g., reviews, average rating, installs [J1313]. Further, malicious developers also attempt to use app markets as tools to widely distribute their malware apps. We conjecture that a longitudinal analysis of apps can reveal both fraudulent and malicious apps. In the following we provide supporting evidence.

Search rank fraud. We have contacted Freelancer workers specializing in Google Play fraud, and have obtained the ids of 2,600 Google Play accounts that were used to write fraudulent reviews for 201 unique apps. We have analyzed these apps and found that fraudulent app search optimization attempts often produce suspicious review patterns. A longitudinal analysis of an app's reviews, which we call *timeline*, can reveal such patterns. For instance, Figure 3.10(a) shows the review timeline of "Daily Yoga- Yoga Fitness Plans", one of the 201 apps targeted by the 15 fraudster-controlled accounts. We observe several suspicious positive reviews spikes, some at over 200 reviews per day, in contrast with long intervals of under 50 daily positive reviews.

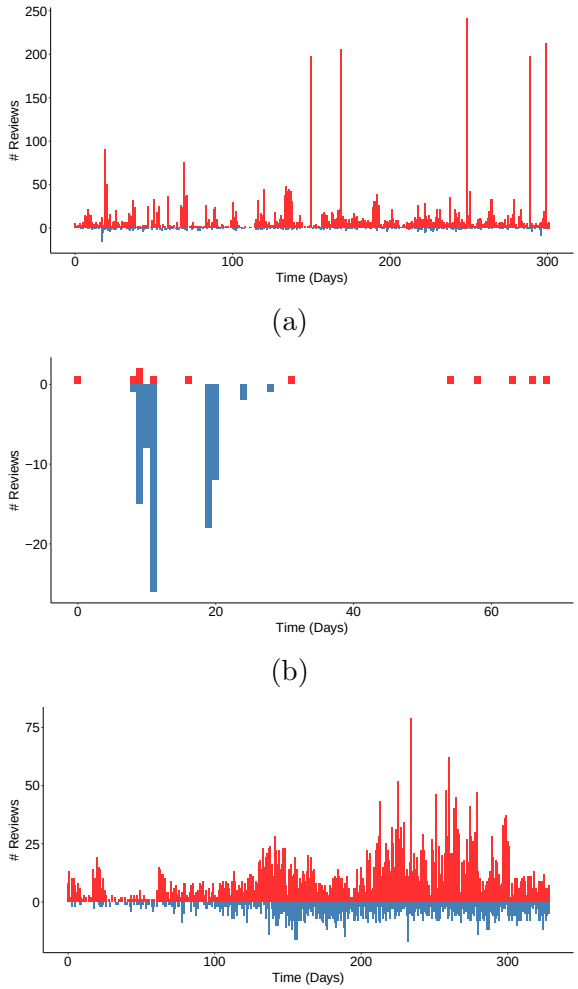


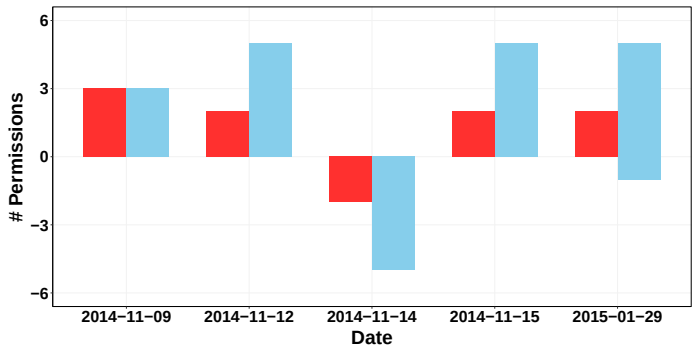
Figure 3.10: Review timeline of fraud apps: x axis shows time with a day granularity, y axis the number of daily positive reviews (red, positive direction) and negative reviews (blue, negative direction). Apps can be targets of both positive and negative search rank fraud campaigns: (a) The app Daily Yoga- Yoga Fitness Plans had days with above 200 positive review spikes. (b) Real Caller received suspicious negative review spikes from ground truth fraudster-controlled accounts. (c) Crownit - Cashback & Prizes received both positive and negative reviews from fraudster-controlled accounts.

We have observed that Google Play apps can also be the target of negative review campaigns, receiving negative reviews from multiple fraudster-controlled accounts. Figure 3.10(b) shows the timeline of such an app, "Real Caller", where we observe days with up to 25 negative reviews, but few positive reviews. While negative reviews are often associated with poor quality apps, these particular spikes are generated from the fraudster-controlled accounts mentioned above. We conjecture that negative review campaigns are sponsored by competitors. Further, we identified apps that are the target of both positive and negative reviews. Figure 3.10(c) shows the timeline of such an app, "Crownit - Cashback & Prizes". While the app has received more positive reviews with higher spikes, its negative reviews and spikes thereof are also significant.

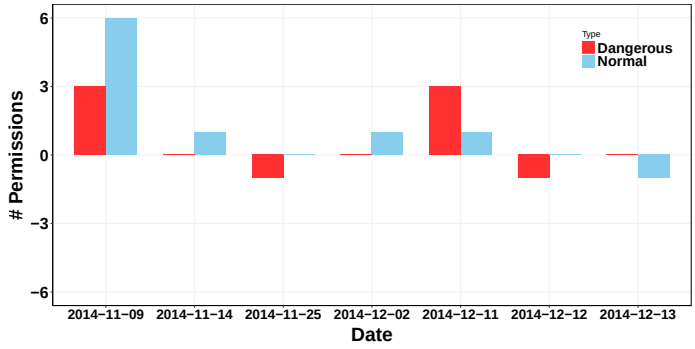
App markets can monitor timelines and notify developers and their users when such suspicious spikes occur.

In addition, our analysis has shown that several developers upload many unpopular apps (see §3.6), while others tend to push frequent updates (§3.5). We describe here vulnerabilities related to such behaviors.

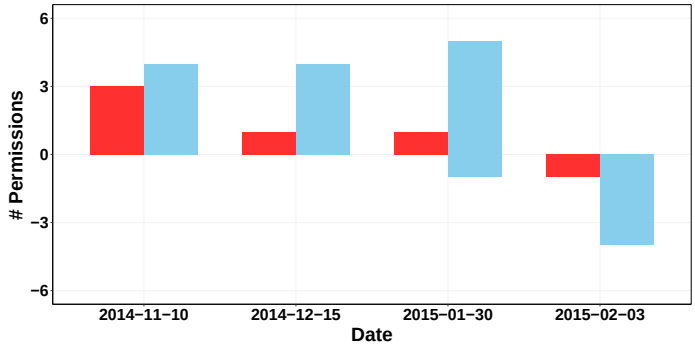
Scam Apps. We have identified several "productive" developers, that upload many similar apps. Among them, we have observed several thousands of premium applications (priced around $1.99) that are slight variations of each other and have almost no observable functionality. Such apps rely on their names and description to scam users into paying for them, then fail to deliver. Each such app receives ≈500-1000 downloads, bringing its developer a profit of $1000-2000.



(a)



(b)



(c)

Figure 3.11: Permission timeline of 3 VirusTotal flagged apps, (a) Hidden Object Blackstone, (b) Top Race Manager, and (c) Cash Yourself. The x axis shows the date when the permission changes occurred; the y axis shows the number of permissions that were newly requested (positive direction) or removed (negative direction). The red bars show dangerous permissions, blue bars regular permissions. We observe significant permission changes, even within days.

Malware. While updates enable developers to fix bugs and push new functionality in a seamless manner, attack vectors can also leverage them. Such attack vectors can be exploited both by malicious developers and by attackers that infiltrate developer accounts. We posit that a motivated attacker can develop and upload a benign app, and once it gains popularity, push malware as an update.

On the iOS platform, Wang et al. [WLL+13] proposed to make the app remotely exploitable, then introduce malicious control flows by rearranging already signed code. We propose an Android variant where the attacker ramps up the permissions required by the app, exploiting the observation that a user is more likely to accept them, then to uninstall the app.

To provide an intuition behind our conjecture, we introduce the concept of *app permission timeline*, the evolution in time of an app's requests for new permissions, or decisions to remove permissions. We have used VirusTotal [Vir15] to test the apks of 7,756 randomly selected apps from the dataset.14-15. We have selected apps for which VirusTotal raised at least 3 flags and that have at least 10 reviews. Figure 3.11 shows the permission timeline of 3 of these apps, for both dangerous (red bars) and regular permissions (blue bars).

For instance, the "Hidden Object Blackstone" app (Figure 3.11(a)) has a quick succession of permission requests and releases at only a few days apart. While the app releases 2 dangerous permissions on November 14, 2014, it requests them again 1 day later, and requests 2 more a month and a half later. Similarly, the "Top Race Manager" app (Figure 3.11(b)) has very frequent permission changes, daily for the last 3. The "Cash Yourself" app (Figure 3.11(c)) requests 3 dangerous permissions on November 10 2014, followed by 1 dangerous permission in both December and January, then releases 1 dangerous permission 4 days later.

Permission changes imply significant app changes. Frequent and significant permission changes, especially the dangerous ones may signal malware, or unstable apps.

Market owners can decide to carefully scan the updates of such apps for malware, and notify developers that something went wrong with their updates, indicating potential account infiltration.

3.7.2 App Market Ecosystem

Analytics-driven Application Development. We envision a development model where insights derived from raw market-level data is integrated into the application development. Such a model is already adopted by websites such as Priceline [pri] through their "Name Your Own Price" scheme where the interface provides users with hints on setting an optimal price towards a successful bid. We propose the extension of development tools like Google's Android Studio [and] with market-level analytics, including:

- **Median price**: In §3.5.1, we showed that developers may be settling down for lower profits. The development tools could provide developers them with hints on the optimal price for their app based on, e.g., the number of features, the price of active apps in the same category etc.
- **Application risk**: Provide predictions on the impact of permissions and updates on reviews and download count.
- **App insights**: Present actionable insights extracted from user reviews (e.g., using solutions like NetSieve [PJNR13]), including most requested feature, list of buggy features, features that crash the app.

Enriching User Experience. We believe data-driven insights will be indispensable to enhance the end user experience:

- **Analytics based app choice**: Visualize app price, update overhead, required permissions, reviewer sentiment to enhance the user experience when choosing among apps with similar claimed functionality. For instance, develop scores for

individual features, and even an overall "sorting" score based on user preferences. Scam apps (see §3.7.1) should appear at the bottom of the score based sorted app list.

- **Analytics based app quarantine**: We envision a quarantine based approach to defend against "update" attacks. An update installation is postponed until analytics of variation in app features indicates the update is stable and benign. To avoid a situation where all users defer installation, we propose a probabilistic quarantine. Each user can update the app after a personalized random interval after its release.

3.8 Limitations

This paper seeks to shed light on the dynamics of the Google app market and also provide evidence that a longitudinal monitoring of apps is beneficial for users, app developers and the market owners. However, our datasets were collected in 2012 and 2014-2015, and may not reflect the current trends of Google Play.

In addition, while we believe that the Google Play market, the applications it hosts and developers we examined represent a large body of other third-party markets and their environments, we do not intend to generalize our results to all the smartphone markets. The characteristics and findings obtained in this study are associated with the Google Play market and its developers. Therefore, the results should be taken with the market and our data collection methodology in mind.

The goal of our discussion of permission and review timelines was to provide early evidence that a longitudinal monitoring and analysis of apps in app markets can be used to identify suspicious apps. We leave for future work a detailed study of permission changes to confirm their statistical significance in detecting search rank fraud and malware.

3.9 Research Contributions Acknowledgment

It is a pleasure to acknowledge the contributions of my co-authors; Dr. Rahul Potharaju and my supervisor Dr. Bogdan Carbunar for their critical remarks on this research and helping me out of a technical spot. Dr. Rahul Potharaju has designed the initial framework for the Google Play crawler, performed longitudinal monitoring and analyzed dataset.2012 to provide valuable intellectual insights. Dr. Bogdan Carbunar contributed a lot to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript.

3.10 Summary

This article studies temporal patterns in Google Play, an influential app market. We use data we collected from more than 160,000 apps daily over a six month period, to examine market trends, application characteristics and developer behavior in real-world market settings. Our work provides insights into the impact of developer levers (e.g., price, permissions requested, update frequency) on app popularity. We proposed future directions for integrating analytics insights into developer and user experiences. We introduced novel attack vectors on app markets and discussed future detection directions.

CHAPTER 4

Search Rank Fraud and Malware Detection

4.1 Motivation

The commercial success of Android app markets such as Google Play [Goob] and the incentive model they offer to popular apps, make them appealing targets for fraudulent and malicious behaviors. Some fraudulent developers deceptively boost the search rank and popularity of their apps (e.g., through fake reviews and bogus installation counts) [Sie14], while malicious developers use app markets as a launch pad for their malware [Min14, Mlo14, Rob15, Gre14]. The motivation for such behaviors is impact: app popularity surges translate into financial benefits and expedited malware proliferation.

Fraudulent developers frequently exploit crowdsourcing sites (e.g., Freelancer [Fre], Fiverr [Fiv], BestAppPromotion [Bes]) to hire teams of willing workers to commit fraud collectively, emulating realistic, spontaneous activities from unrelated people (i.e., "crowdturfing" [WWZ+12]).

In addition, the efforts of Android markets to identify and remove malware are not always successful. For instance, Google Play uses the Bouncer system [OM12] to remove malware. However, out of the 7,756 Google Play apps we analyzed using VirusTotal [Vir15], 12% (948) were flagged by at least one anti-virus tool and 2% (150) were identified as malware by at least 10 tools (see Figure 4.5).

Previous mobile malware detection work has focused on dynamic analysis of app executables [BZNT11, SKE+12, GZZ+12] as well as static analysis of code and permissions [SLG+12, PGS+12, YSM14]. However, recent Android malware analysis revealed that malware evolves quickly to bypass anti-virus tools [ZJ12].

In this paper, we seek to identify both malware and search rank fraud subjects in Google Play. This combination is not arbitrary: we posit that malicious developers resort to search rank fraud to boost the impact of their malware.

Unlike existing solutions, we build this work on the observation that fraudulent and malicious behaviors leave behind telltale signs on app markets. We uncover these nefarious acts by picking out such trails. For instance, the high cost of setting up valid Google Play accounts forces fraudsters to reuse their accounts across review writing jobs, making them likely to review more apps in common than regular users. Resource constraints can compel fraudsters to post reviews within short time intervals. Legitimate users affected by malware may report unpleasant experiences in their reviews. Increases in the number of requested permissions from one version to the next, which we will call "permission ramps", may indicate benign to malware (Jekyll-Hyde) transitions.

4.1.1 Contributions

We propose FairPlay, a system that leverages the above observations to efficiently detect Google Play fraud and malware (see Figure 4.6). Our major contributions are:
A fraud and malware detection approach. To detect fraud and malware, we propose and generate 28 relational, behavioral and linguistic features, that we use to train supervised learning algorithms [§ 4.4]:

- We formulate the notion of *co-review graphs* to model reviewing relations between users. We develop PCF, an efficient algorithm to identify temporally constrained, co-review pseudo-cliques — formed by reviewers with substantially overlapping co-reviewing activities across short time windows.
- We use temporal dimensions of review post times to identify suspicious review spikes received by apps; we show that to compensate for a negative review, for

an app that has rating R, a fraudster needs to post at least $\frac{R-1}{5-R}$ positive reviews. We also identify apps with "unbalanced" review, rating and install counts, as well as apps with permission request ramps.

- We use linguistic and behavioral information to (i) detect genuine reviews from which we then (ii) extract user-identified fraud and malware indicators.

Tools to collect and process Google Play data. We have developed GPCrawler, a tool to automatically collect data published by Google Play for apps, users and reviews, as well as GPad, a tool to download apks of free apps and scan them for malware using VirusTotal.

Novel longitudinal and gold standard datasets. We contributed a longitudinal dataset of 87,223 freshly posted Google Play apps (along with their 2.9M reviews, from 2.3M reviewers) collected between October 2014 and May 2015. We have leveraged search rank fraud expert contacts in Freelancer [Fre], anti-virus tools and manual verifications to collect gold standard datasets of hundreds of fraudulent, malware and benign apps [§ 5.6.1]. We have published these datasets on the project website [Soc].

4.1.2 Results

FairPlay has high accuracy and real-world impact:

High Accuracy. FairPlay achieves over 97% accuracy in classifying fraudulent and benign apps, and over 95% accuracy in classifying malware and benign apps. FairPlay significantly outperforms the malware indicators of Sarma et al. [SLG+12]. Furthermore, we show that malware often engages in search rank fraud as well: When trained on fraudulent and benign apps, FairPlay flagged as fraudulent more than 75% of the gold standard malware apps [§ 4.5.3].

Real-world Impact: Uncover Fraud & Attacks. FairPlay discovers hundreds of fraudulent apps. We show that these apps are indeed suspicious: the reviewers of

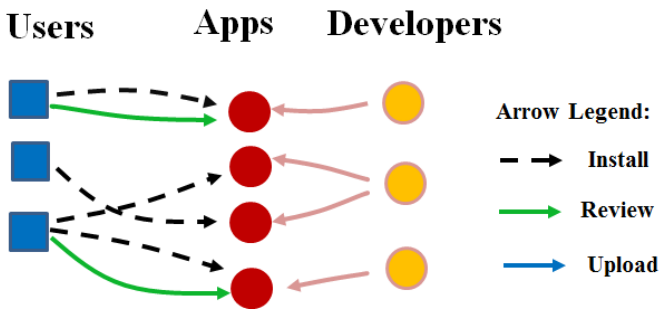


Figure 4.1: Google Play components and relations. Google Play's functionality centers on apps, shown as red disks. Developers, shown as orange disks upload apps. A developer may upload multiple apps. Users, shown as blue squares, can install and review apps. A user can only review an app that he previously installed.

93.3% of them form at least 1 pseudo-clique, 55% of these apps have at least 33% of their reviewers involved in a pseudo-clique, and the reviews of around 75% of these apps contain at least 20 words indicative of fraud.

FairPlay also enabled us to discover a novel, *coercive review campaign* attack type, where app users are harassed into writing a positive review for the app, and install and review other apps. We have discovered $1,024$ coerced reviews, from users complaining about 193 such apps [§ 4.5.4 & § 4.5.5].

4.2 Background, Related Work, and Our Differences

System model. We focus on the Android app market ecosystem of Google Play. The participants, consisting of users and developers, have Google accounts. Developers create and upload apps, that consist of executables (i.e., "apks"), a set of required permissions, and a description. The app market publishes this information, along with the app's received reviews, ratings, aggregate rating (over both reviews and ratings), install count range (predefined buckets, e.g., 50-100, 100-500), size, version number, price, time of last update, and a list of "similar" apps. Each review consists of a star

rating ranging between 1-5 stars, and some text. The text is optional and consists of a title and a description. Google Play limits the number of reviews displayed for an app to $4,000$. Figure 4.1 illustrates the participants in Google Play and their relations.

Adversarial model. We consider not only malicious developers, who upload malware, but also rational fraudulent developers. Fraudulent developers attempt to tamper with the search rank of their apps, e.g., by recruiting fraud experts in crowdsourcing sites to write reviews, post ratings, and create bogus installs. While Google keeps secret the criteria used to rank apps, the reviews, ratings and install counts are known to play a fundamental part (see e.g., [Ank13]).

To review or rate an app, a user needs to have a Google account, register a mobile device with that account, and install the app on the device. This process complicates the job of fraudsters, who are thus more likely to reuse accounts across jobs. The reason for search rank fraud attacks is impact. Apps that rank higher in search results, tend to receive more installs. This is beneficial both for fraudulent developers, who increase their revenue, and malicious developers, who increase the impact of their malware.

4.3 The Data

We have collected longitudinal data from 87K+ newly released apps over more than 6 months, and identified gold standard data. In the following, we briefly describe the tools we developed, then detail the data collection effort and the resulting datasets.

Data collection tools. We have developed the *Google Play Crawler* (GPCrawler) tool, to automatically collect data published by Google Play for apps, users and reviews. Google Play prevents scripts from scrolling down a user page. Thus, to collect the ids of more than 20 apps reviewed by a user. To overcome this limitation,

we developed a Python script and a Firefox add-on. Given a user id, the script opens the user page in Firefox. When the script loads the page, the add-on becomes active. The add-on interacts with Google Play pages using content scripts (Browser specific components that let us access the browsers native API) and port objects for message communication. The add-on displays a "scroll down" button that enables the script to scroll down to the bottom of the page. The script then uses a DOMParser to extract the content displayed in various formats by Google Play. It then sends this content over IPC to the add-on. The add-on stores it, using Mozilla XPCOM components, in a sand-boxed environment of local storage in a temporary file. The script then extracts the list of apps rated or reviewed by the user.

We have also developed the *Google Play App Downloader* (GPad), a Java tool to automatically download apks of free apps on a PC, using the open-source *Android Market API* [And11]. GPad takes as input a list of free app ids, a Gmail account and password, and a GSF id. GPad creates a new market session for the "androidsecure" service and logs in. GPad sets parameters for the session context (e.g., mobile device Android SDK version, mobile operator, country), then issues a *GetAssetRequest* for each app identifier in the input list. GPad introduces a 10s delay between requests. The result contains the url for the app; GPad uses this url to retrieve and store the app's binary stream into a local file. After collecting the binaries of the apps on the list, GPad scans each app apk using VirusTotal [Vir15], an online malware detector provider, to find out the number of anti-malware tools (out of 57: AVG, McAfee, Symantec, Kaspersky, Malwarebytes, F-Secure, etc.) that identify the apk as suspicious. We used 4 servers (PowerEdge R620, Intel Xeon E-26XX v2 CPUs) to collect our datasets, which we describe next.

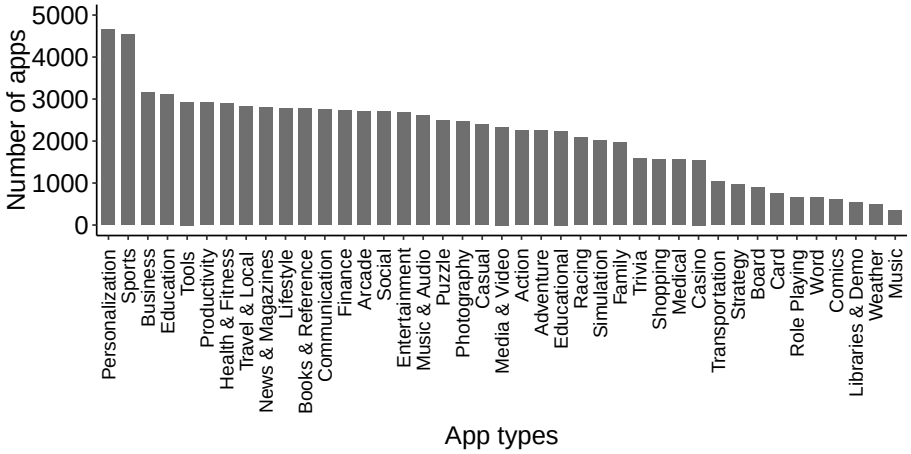


Figure 4.2: Distribution of app types for the 87,223 fresh app set. With the slight exception of Personalization and Sports type spikes, we have achieved an almost uniform distribution across all app types, as desirable.

4.3.1 Longitudinal App Data

In order to detect suspicious changes that occur early in the lifetime of apps, we used the "New Releases" link to identify apps with a short history on Google Play. Our interest in newly released apps stems from our analysis of search rank fraud jobs posted on crowdsourcing sites, that revealed that app developers often recruit fraudsters early after uploading their apps on Google Play. Their intent is likely to create the illusion of an up-and-coming app, that may then snowball with interest from real users. By monitoring new apps, we aim to capture in real-time the moments when such search rank fraud campaigns begin.

We approximate the first upload date of an app using the day of its first review. We have started collecting new releases in July 2014 and by October 2014 we had a set of 87,223 apps, whose first upload time was under 40 days prior to our first collection time, when they had at most 100 reviews.

Figure 4.2 shows the distribution of the fresh app categories. We have collected app from each category supported by Google Play, with at least 500 apps per category (Music & Audio) and more than 4,500 for the most popular category (Personaliza-

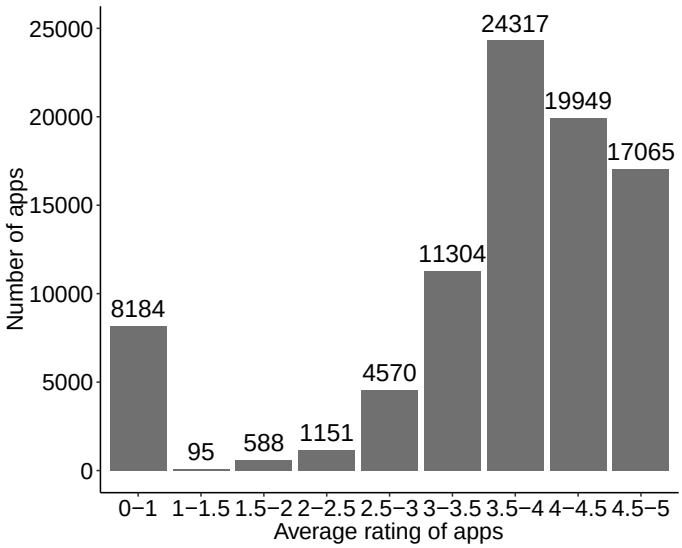


Figure 4.3: Average rating distribution for the 87, 223 fresh app set. Most apps have more than 3.5 stars, few have between 1 and 2.5 stars, but more than 8, 000 apps have less than 1.

tion). Figure 4.3 shows the average rating distribution of the fresh apps. Most apps have at least a 3.5 rating aggregate rating, with few apps between 1 and 2.5 stars. However, we observe a spike at more than 8, 000 apps with less than 1 star.

We have collected longitudinal data from these 87, 223 apps between October 24, 2014 and May 5, 2015. Specifically, for each app we captured "snapshots" of its Google Play metadata, twice a week. An app snapshot consists of values for all its time varying variables, e.g., the reviews, the rating and install counts, and the set of requested permissions (see § 4.2 for the complete list). For each of the 2, 850, 705 reviews we have collected from the 87, 223 apps, we recorded the reviewer's name and id (2, 380, 708 unique ids), date of review, review title, text, and rating.

This app monitoring process enables us to extract a suite of unique features, that include review, install and permission changes. In particular, we note that this approach enables us to overcome the Google Play limit of 4000 displayed reviews per app: each snapshot will capture only the reviews posted after the previous snapshot.

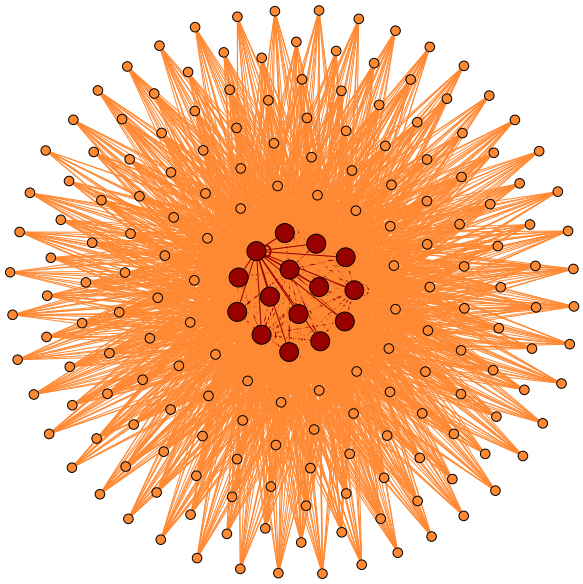


Figure 4.4: Co-review graph of 15 seed fraud accounts (red nodes) and the 188 GbA accounts (orange nodes). Edges indicate reviews written in common by the accounts corresponding to the endpoints. We only show edges having at least one seed fraud account as an endpoint. The 15 seed fraud accounts form a suspicious clique with edges weights that range between 60 and 217. The GbA accounts are also suspiciously well connected to the seed fraud accounts: the weights of their edges to the seed fraud accounts ranges between 30 and 302.

4.3.2 Gold Standard Data

Malware apps. We used GPad (see § 5.6.1) to collect the apks of 7,756 randomly selected apps from the longitudinal set (see § 4.3.1). Figure 4.5 shows the distribution of flags raised by VirusTotal, for the 7,756 apks. None of these apps had been filtered by Bouncer [OM12]! From the 523 apps that were flagged by at least 3 tools, we selected those that had at least 10 reviews, to form our "malware app" dataset, for a total of 212 apps. We collected all the 8,255 reviews of these apps.

Fraudulent apps. We used contacts established among Freelancer [Fre]'s search rank fraud community, to obtain the identities of 15 Google Play accounts that were used to write fraudulent reviews for 201 unique apps. We call the 15 accounts "seed fraud accounts" and the 201 apps "seed fraud apps". Figure 4.4 shows the graph formed by the review habits of the 15 seed accounts: nodes are accounts, edges connect

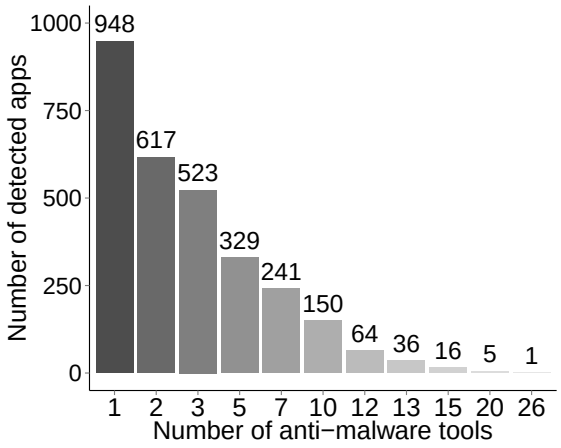


Figure 4.5: Apks detected as suspicious (y axis) by multiple anti-virus tools (x axis), through VirusTotal [Vir15], from a set of 7,756 downloaded apks.

accounts who reviewed apps in common, and edge weights represent the number of such commonly reviewed apps. The 15 seed fraud accounts form a suspicious clique. This shows that worker controlled accounts are used to review many apps in common: the weights of the edges between the seed fraud accounts range between 60 and 217.

Fraudulent reviews. We have collected all the 53,625 reviews received by the 201 seed fraud apps. The 15 seed fraud accounts were responsible for 1,969 of these reviews. We used the 53,625 reviews to identify 188 accounts, such that each account was used to review at least 10 of the 201 seed fraud apps (for a total of 6,488 reviews). We call these, *guilt by association* (GbA) accounts. Figure 4.4 shows the co-review edges between these GbA accounts (in orange) and the seed fraud accounts: the GbA accounts are suspiciously well connected to the seed fraud accounts, with the weights of their edges to the seed accounts ranging between 30 and 302.

To reduce feature duplication, we have used the 1,969 fraudulent reviews written by the 15 seed accounts and the 6,488 fraudulent reviews written by the 188 GbA accounts for the 201 seed fraud apps, to extract a *balanced* set of fraudulent reviews. Specifically, from this set of 8,457 ($= 1,969 + 6,488$) reviews, we have collected 2 reviews from each of the 203 ($= 188 + 15$) suspicious user accounts. Thus, the gold

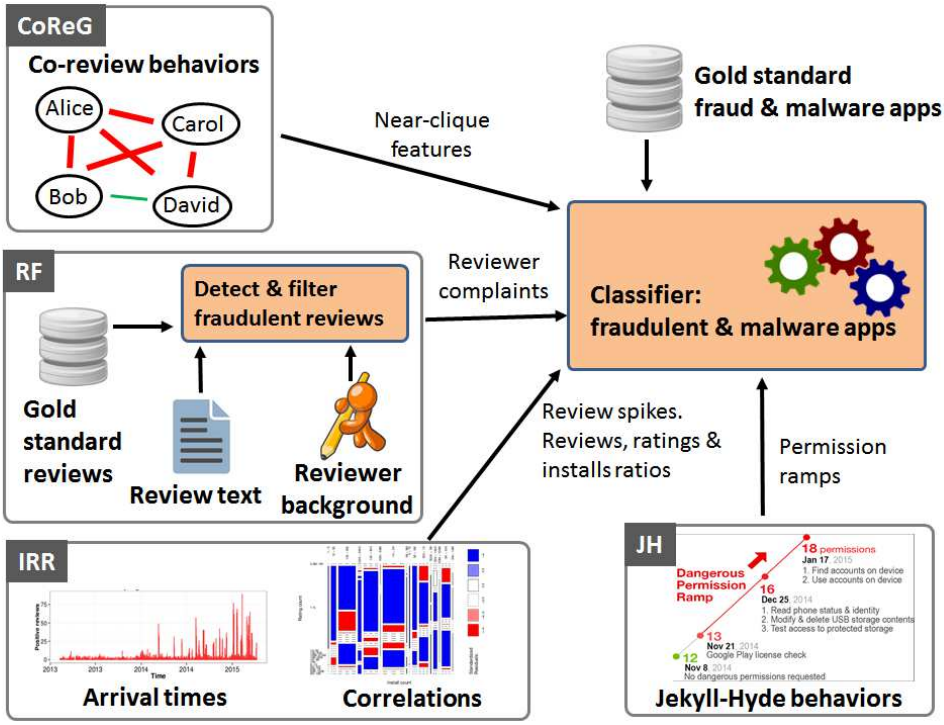


Figure 4.6: FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.

Notation	Definition
CoReG Module	
$nCliques$	number of pseudo-cliques with $\rho \geq \theta$
ρ_{max}, ρ_{med}, ρ_{SD}	clique density: max, median, SD
CS_{max}, CS_{med}, CS_{SD}	pseudo-cliques size: max, median, SD
$inCliqueCount$	% of nodes involved in pseudo-cliques
RF Module	
$malW$	% of reviews with malware indicators
$fraudW$, $goodW$	% of reviews with fraud/benign words
FRI	fraud review impact on app rating
IRR Module	
$spikeCount$, $spike_{amp}$	days with spikes & spike amplitude
I_1/Rt_1, I_2/Rt_2	install to rating ratios
I_1/Rv_1, I_2/Rv_2	install to review ratios
JH Module	
$permCt$, $dangerCount$	# of total and dangerous permissions
$rampCt$	# of dangerous permission ramps
$dangerRamp$	# of dangerous permissions added

Table 4.1: FairPlay's most important features, organized by their extracting module. § 4.4.2 describes ρ and θ.

standard dataset of fraudulent reviews consists of 406 reviews.

The reason for collecting a small number of reviews from each fraudster is to reduce feature duplication: many of the features we use to classify a review are extracted from the user who wrote the review (see Table 4.2).

Benign apps. We have selected 925 candidate apps from the longitudinal app set, that have been developed by Google designated "top developers". We have used GPad to filter out those flagged by VirusTotal. We have manually investigated 601 of the remaining apps, and selected a set of 200 apps that (i) have more than 10 reviews and (ii) were developed by reputable media outlets (e.g., NBC, PBS) or have an associated business model (e.g., fitness trackers). We have also collected the $32,022$ reviews of these apps.

Genuine reviews. We have manually collected a gold standard set of 315 genuine reviews, as follows. First, we have collected the reviews written for apps installed on

the Android smartphones of the authors. We then used Google's text and reverse image search tools to identify and filter those that plagiarized other reviews or were written from accounts with generic photos. We have then manually selected reviews that mirror the authors' experience, have at least 150 characters, and are informative (e.g., provide information about bugs, crash scenario, version update impact, recent changes).

4.4 FairPlay: Proposed Solution

We now introduce FairPlay, a system to automatically detect malicious and fraudulent apps.

4.4.1 FairPlay Overview

FairPlay organizes the analysis of longitudinal app data into the following 4 modules, illustrated in Figure 4.6. The Co-Review Graph (CoReG) module identifies apps reviewed in a contiguous time window by groups of users with significantly overlapping review histories. The Review Feedback (RF) module exploits feedback left by genuine reviewers, while the Inter Review Relation (IRR) module leverages relations between reviews, ratings and install counts. The Jekyll-Hyde (JH) module monitors app permissions, with a focus on dangerous ones, to identify apps that convert from benign to malware. Each module produces several features that are used to train an app classifier. FairPlay also uses general features such as the app's average rating, total number of reviews, ratings and installs, for a total of 28 features. Table 4.1 summarizes the most important features. We now detail each module and the features it extracts.

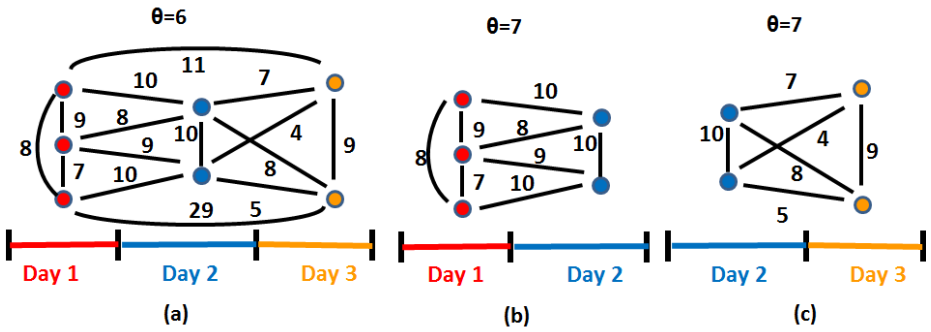


Figure 4.7: Example pseudo-cliques and PCF output. Nodes are users and edge weights denote the number of apps reviewed in common by the end users. Review timestamps have a 1-day granularity. (a) The entire co-review graph, detected as pseudo-clique by PCF when θ is 6. When θ is 7, PCF detects the subgraphs of (b) the first two days and (c) the last two days. When θ=8, PCF detects only the clique formed by the first day reviews (the red nodes).

4.4.2 The Co-Review Graph (CoReG) Module

This module exploits the observation that fraudsters who control many accounts will re-use them across multiple jobs. Its goal is then to detect sub-sets of an app's reviewers that have performed significant common review activities in the past. In the following, we describe the co-review graph concept, formally present the weighted maximal clique enumeration problem, then introduce an efficient heuristic that leverages natural limitations in the behaviors of fraudsters.

Co-review graphs. Let the co-review graph of an app, see Figure 4.7, be a graph where nodes correspond to user accounts who reviewed the app, and undirected edges have a weight that indicates the number of apps reviewed in common by the edge's endpoint users. Figure 4.15(a) shows the co-review clique of one of the seed fraud apps (see § 4.3.2). The co-review graph concept naturally identifies user accounts with significant past review activities.

The weighted maximal clique enumeration problem. Let $G = (V, E)$ be a graph, where V denotes the sets of vertices of the graph, and E denotes the set of edges. Let w be a weight function, $w : E \rightarrow \mathbb{R}$ that assigns a weight to each edge of

G. Given a vertex sub-set $U \in V$, we use $G[U]$ to denote the sub-graph of G induced by U. A vertex sub-set U is called a *clique* if any two vertices in U are connected by an edge in E. We say that U is a *maximal clique* if no other clique of G contains U. The *weighted maximal clique enumeration* problem takes as input a graph G and returns the set of maximal cliques of G.

Algorithm 1 PCF algorithm pseudo-code.

```
Input: days, an array of daily reviews, and
       θ, the weighted threshold density
Output: allCliques, set of all detected pseudo-cliques
1. for d :=0  d < days.size(); d++
2.    Graph PC := new Graph();
3.    bestNearClique(PC, days[d]);
4.    c := 1; n := PC.size();
5.    for nd := d+1; d < days.size() & c = 1; d++
6.      bestNearClique(PC, days[nd]);
7.      c := (PC.size() > n); endfor
8.    if (PC.size() > 2)
9.      allCliques := allCliques.add(PC); fi endfor
10. return
11. function bestNearClique(Graph PC, Set revs)
12.   if (PC.size() = 0)
13.     for root := 0; root < revs.size(); root++
14.       Graph candClique := new Graph ();
15.       candClique.addNode (revs[root].getUser());
16.       do candNode := getMaxDensityGain(revs);
17.         if (density(candClique ∪ {candNode}) ≥ θ))
18.           candClique.addNode(candNode); fi
19.       while (candNode != null);
20.       if (candClique.density() > maxRho)
21.           maxRho := candClique.density();
22.           PC := candClique; fi endfor
23.   else if (PC.size() > 0)
24.       do candNode := getMaxDensityGain(revs);
25.         if (density(candClique ∪ candNode) ≥ θ))
26.           PC.addNode(candNode); fi
27.       while (candNode != null);
28. return
```

Maximal clique enumeration algorithms such as [TTT06, MU04] applied to co-review graphs are not ideal to solve the problem of identifying sub-sets of an app's reviewers with significant past common reviews. First, fraudsters may not consistently use (or may even purposefully avoid using) all their accounts across all fraud jobs that they perform. In addition, Google Play provides incomplete information (up to 4,000 reviews per app, may also detect and filter fraud). Since edge information may be incomplete, original cliques may now also be incomplete. To address this problem, we "relax" the clique requirement and focus instead of pseudo-cliques:

The weighted pseudo-clique enumeration problem. For a graph $G = (V, E)$ and a threshold value θ, we say that a vertex sub-set U (and its induced sub-graph $G[U]$) is a *pseudo-clique* of G if its weighted density $\rho = \frac{\sum_{e \in E} w(e)}{\binom{n}{2}}$ [Uno07] exceeds θ; $n = |V|$ [1]. U is a *maximal pseudo-clique* if in addition, no other pseudo-clique of G contains U. The weighted pseudo-clique enumeration problem outputs all the vertex sets of V whose induced subgraphs are weighted pseudo-cliques of G.

The Pseudo Clique Finder (PCF) algorithm. We propose PCF (Pseudo Clique Finder), an algorithm that exploits the observation that fraudsters hired to review an app are likely to post those reviews within relatively short time intervals (e.g., days). PCF (see Algorithm 1), takes as input the set of the reviews of an app, organized by days, and a threshold value θ. PCF outputs a set of identified pseudo-cliques with $\rho \geq \theta$, that were formed during contiguous time frames. In Section 4.5.3 we discuss the choice of θ.

For each day when the app has received a review (line 1), PCF finds the day's most promising pseudo-clique (lines 3 and $12 - 22$): start with each review, then greedily add other reviews to a candidate pseudo-clique; keep the pseudo clique (of the day) with the highest density. With that "work-in-progress" pseudo-clique, move

[1] ρ is thus the average weight of the graph's edges, normalized by the total number of edges of a perfect clique of size n.

on to the next day (line 5): greedily add other reviews while the weighted density of the new pseudo-clique equals or exceeds θ (lines 6 and $23-27$). When no new nodes have been added to the work-in-progress pseudo-clique (line 8), we add the pseudo-clique to the output (line 9), then move to the next day (line 1). The greedy choice ($getMaxDensityGain$, not depicted in Algorithm 1) picks the review not yet in the work-in-progress pseudo-clique, whose writer has written the most apps in common with reviewers already in the pseudo-clique. Figure 4.7 illustrates the output of PCF for several θ values.

If d is the number of days over which A has received reviews and r is the maximum number of reviews received in a day, PCF's complexity is $O(dr^2(r+d))$.

We note that if multiple fraudsters target an app in the same day, PCF may detect only the most densely connected pseudo-clique, corresponding to the most prolific fraudster, and miss the lesser dense ones.

CoReG features. CoReG extracts the following features from the output of PCF (see Table 4.1) (i) the number of cliques whose density equals or exceeds θ, (ii) the maximum, median and standard deviation of the densities of identified pseudo-cliques, (iii) the maximum, median and standard deviation of the node count of identified pseudo-cliques, normalized by n (the app's review count), and (iv) the total number of nodes of the co-review graph that belong to at least one pseudo-clique, normalized by n.

4.4.3 Reviewer Feedback (RF) Module

Reviews written by genuine users of malware and fraudulent apps may describe negative experiences. The RF module exploits this observation through a two step approach: (i) detect and filter out fraudulent reviews, then (ii) identify malware and fraud indicative feedback from the remaining reviews.

Notation	Definition
ρ_R	The rating of R
$L(R)$	The length of R
$pos(R)$	Percentage of positive statements in R
$neg(R)$	Percentage of negative statements in R
$nr(U)$	The number of reviews written by U
$\pi(\rho_R)$	Percentile of ρ_R among all reviews of U
$Exp_U(A)$	The expertise of U for app A
$B_U(A)$	The bias of U for A
$Paid(U)$	The money spent by U to buy apps
$Rated(U)$	Number of apps rated by U
$plusOne(U)$	Number of apps +1'd by U
$n.flwrs(U)$	Number of followers of U in Google+

Table 4.2: Features used to classify review R written by user U for app A.

Step RF.1: Fraudulent review filter. We posit that certain features can accurately pinpoint genuine and fake reviews. We propose several such features, see Table 4.2 for a summary, defined for a review R written by user U for an app A.

• *Reviewer based features.* The *expertise* of U for app A, defined as the number of reviews U wrote for apps that are "similar" to A, as listed by Google Play (see § 4.2). The *bias* of U towards A: the number of reviews written by U for other apps developed by A's developer. In addition, we extract the total money paid by U on apps it has reviewed, the number of apps that U has liked, and the number of Google+ followers of U.

• *Text based features.* We used the NLTK library [BKL09] and the Naive Bayes classifier, trained on two datasets: (i) $1,041$ sentences extracted from randomly selected 350 positive and 410 negative Google Play reviews, and (ii) $10,663$ sentences extracted from 700 positive and 700 negative IMDB movie reviews [PLV02]. 10-fold cross validation of the Naive Bayes classifier over these datasets reveals a false negative rate of 16.1% and a false positive rate of 19.65%, for an overall accuracy of 81.74%. We ran a binomial test [McD09] for a given accuracy of p=0.817 over N=1,041 cases using the binomial distribution $binomial(p, N)$ to assess the 95% confidence interval

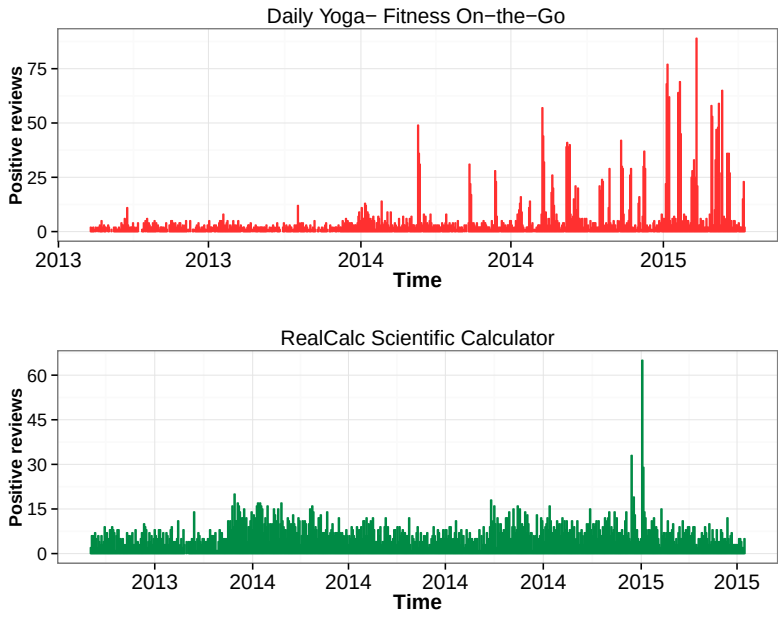


Figure 4.8: Timelines of positive reviews for 2 apps from the fraudulent app dataset. The first app has multiple spikes while the second one has only one significant spike.

for our result. The deviation of the binomial distribution is 0.011. Thus, we are 95% confident that the true performance of the classifier is in the interval (79.55, 83.85).

We used the trained Naive Bayes classifier to determine the statements of R that encode positive and negative sentiments. We then extracted the following features: (i) the percentage of statements in R that encode positive and negative sentiments respectively, and (ii) the rating of R and its percentile among the reviews written by U.

In Section 6.8 we evaluate the review classification accuracy of several supervised learning algorithms trained on these features and on the gold standard datasets of fraudulent and genuine reviews introduced in Section 4.3.2.

Step RF.2: Reviewer feedback extraction. We conjecture that (i) since no app is perfect, a "balanced" review that contains both app positive and negative sentiments is more likely to be genuine, and (ii) there should exist a relation between the review's dominating sentiment and its rating. Thus, after filtering out fraudulent

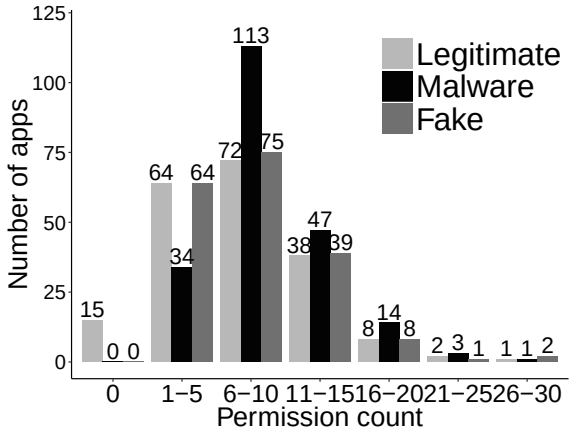
reviews, we extract feedback from the remaining reviews. For this, we have used NLTK to extract 5,106 verbs, 7,260 nouns and 13,128 adjectives from the 97,071 reviews we collected from the 613 gold standard apps (see § 4.3.2). We removed non ascii characters and stop words, then applied lemmatization and discarded words that appear at most once. We have attempted to use stemming, extracting the roots of words, however, it performed poorly. This is due to the fact that reviews often contain (i) shorthands, e.g., "ads", "seeya", "gotcha", "inapp", (ii) misspelled words, e.g., "pathytic", "folish", "gredy", "dispear" and even (iii) emphasized misspellings, e.g., "hackkked", "spammmerrr", "spooooky". Thus, we ignored stemming.

We used the resulting words to manually identify lists of words indicative of malware, fraudulent and benign behaviors. Our malware indicator word list contains 31 words (e.g., risk, hack, corrupt, spam, malware, fake, fraud, blacklist, ads). The fraud indicator word list contains 112 words (e.g., cheat, hideous, complain, wasted, crash) and the benign indicator word list contains 105 words.

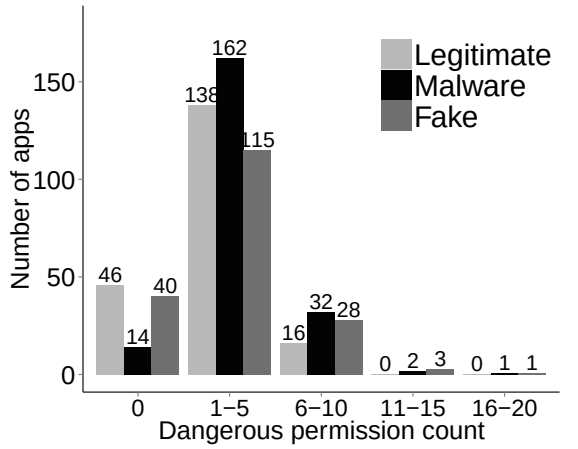
RF features. We extract 3 features (see Table 4.1), denoting the percentage of genuine reviews that contain malware, fraud, and benign indicator words respectively. We also extract the *impact* of detected fraudulent reviews on the overall rating of the app: the absolute difference between the app's average rating and its average rating when ignoring all the fraudulent reviews.

4.4.4 Inter-Review Relation (IRR) Module

This module leverages temporal relations between reviews, as well as relations between the review, rating and install counts of apps, to identify suspicious behaviors.



(a)



(b)

(c)

Figure 4.9: (a) Distribution of total number of permissions requested by malware, fraudulent and legitimate apps. (b) Distribution of the number of "dangerous" permissions requested by malware, fraudulent and benign apps. (c) Dangerous permission ramp during version updates for a sample app "com.battery.plusfree". Originally the app requested no dangerous permissions.

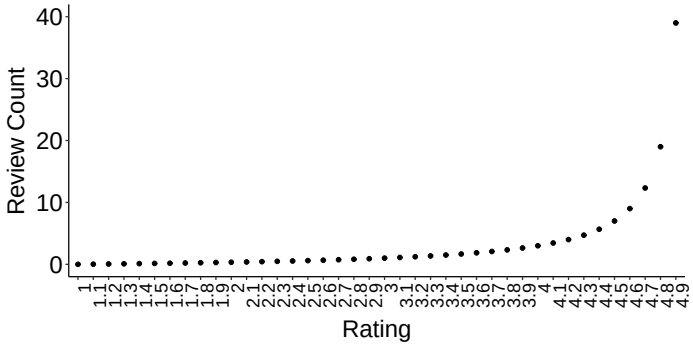


Figure 4.10: Lower bound on the number of fake reviews that need to be posted by an adversary to cancel a 1-star review, vs. the app's current rating (shown with 0.1-star granularity). At 4 stars, the adversary needs to post 3 5-star reviews to cancel a 1-star review, while at 4.2 stars, 4 5-star reviews are needed.

Temporal relations. In order to compensate for a negative review, an attacker needs to post a significant number of positive reviews. Specifically,

Claim 1 *Let R_A denote the average rating of an app A just before receiving a 1 star review. In order to compensate for the 1 star review, an attacker needs to post at least $\frac{R_A-1}{5-R_A}$ positive reviews.*

Proof: Let σ be the sum of all the k reviews received by a before time T. Then, $R_A = \frac{\sigma}{k}$. Let q_r be the number of fraudulent reviews received by A. To compensate for the 1 star review posted at time T, q_r is minimized when all those reviews are 5 star. We then have that: $R_A = \frac{\sigma}{k} = \frac{\sigma+1+5q_r}{k+1+q_r}$. The numerator of the last fraction denotes the sum of all the ratings received by A after time T and the denominator is the total number of reviews. Rewriting the last equality, we obtain that $q_r = \frac{\sigma-k}{5k-\sigma} = \frac{R_A-1}{5-R_A}$. The last equality follows by dividing both the numerator and denominator by k.

Figure 4.10 plots the lower bound on the number of fake reviews that need to be posted to cancel a 1-star review, vs. the app's current rating. It shows that the number of reviews needed to boost the rating of an app is not constant. Instead, as a review campaign boosts the rating of the subject app, the number of fake reviews needed to continue the process, also increases. For instance, a 4 star app needs to

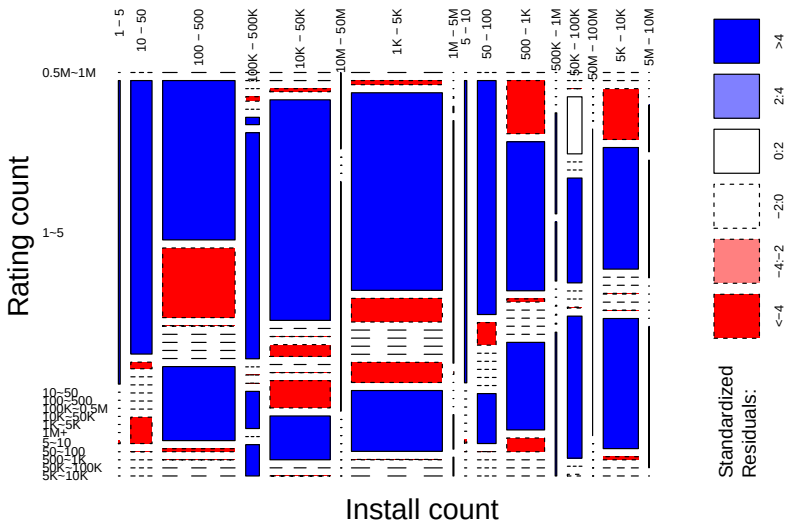


Figure 4.11: Mosaic plot of install vs. rating count relations of the 87K apps. Larger cells (rectangles) signify that more apps have the corresponding rating and install count range; dotted lines mean no apps in a certain install/rating category. The standardized residuals identify the cells that contribute the most to the χ^2 test. The most significant rating:install ratio is 1:100.

receive 3, 5-star reviews to compensate for a single 1 star review, while a 4.2 star app needs to receive 4 such reviews. Thus, adversaries who want to increase the rating of an app, i.e., cancel out previously received negative reviews, will need to post an increasing, significant number of positive reviews.

Such a "compensatory" behavior is likely to lead to suspiciously high numbers of positive reviews. We detect such behaviors by identifying outliers in the number of daily positive reviews received by an app. Figure 4.8 shows the timelines and suspicious spikes of positive reviews for 2 apps from the fraudulent app dataset (see Section 4.3.2). We identify days with spikes of positive reviews as those whose number of positive reviews exceeds the upper outer fence of the box-and-whisker plot built over the app's numbers of daily positive reviews.

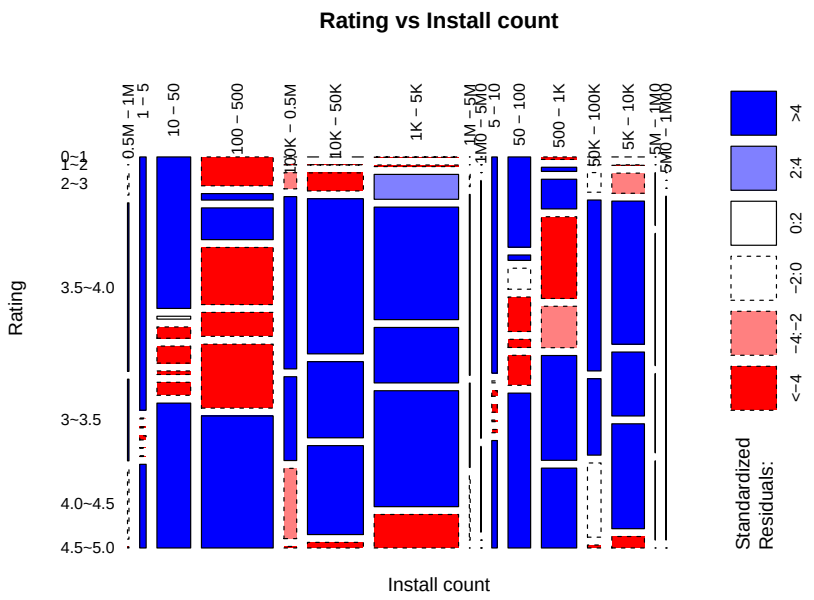


Figure 4.12: Mosaic plot showing relationships between the install count and the average app rating, over the 87K apps. A cell contains the apps that have a certain install count interval (x axis) and rating interval (y axis). Larger cells contain more apps. We observe a relationship between install count and rating: apps that receive more installs also tend to have higher average ratings (i.e., above 3 stars). This may be due to app popularity relationship to quality which may be further positively correlated with app rating.

Reviews, ratings and install counts. We used the Pearson's χ^2 test to investigate relationships between the install count and the rating count, as well as between the install count and the average app rating of the 87K new apps, at the end of the collection interval. We grouped the rating count in buckets of the same size as Google Play's install count buckets. Figure 4.11 shows the mosaic plot of the relationships between rating and install counts. p=0.0008924, thus we conclude dependence between the rating and install counts. The standardized residuals identify the cells (rectangles) that contribute the most to the χ^2 test. The most significant rating:install ratio is 1:100.

In addition, Figure 4.12 shows the mosaic plot of the app install count vs. the average app rating. Rectangular cells correspond to apps that have a certain install count range (x axis) and average rating range (y axis). It shows that few popular apps, i.e., with more than $1,000$ installs, have below 3 stars, or above 4.5 stars. We conjecture that fraudster efforts to alter the search rank of an app will not be able to preserve a natural balance of the features that impact it (e.g., the app's review, rating, and install counts), which will easily be learned and detected by supervised learning algorithms.

IRR features. We extract temporal features (see Table 4.1): the number of days with detected spikes and the maximum amplitude of a spike. We also extract (i) the ratio of installs to ratings as two features, I_1/Rt_1 and I_2/Rt_2 and (ii) the ratio of installs to reviews, as I_1/Rv_1 and I_2/Rv_2. $(I_1, I_2]$ denotes the install count interval of an app, $(Rt_1, Rt_2]$ its rating interval and $(Rv_1, Rv_2]$ its (genuine) review interval.

4.4.5 Jekyll-Hyde App Detection (JH) Module

Figure 4.9(a) shows the distribution of the total number of permissions requested by malware, fraudulent and legitimate apps. Surprisingly, not only malware and fraudulent apps but also legitimate apps request large numbers of permissions.

In addition, Android's API level 22 labels 47 permissions as "dangerous". Figure 4.9(b) compares the distributions of the number of dangerous permissions requested by the gold standard malware, fraudulent and benign apps. The most popular dangerous permissions among these apps are "modify or delete the contents of the USB storage", "read phone status and identity", "find accounts on the device", and "access precise location". Only 8% of the legitimate apps request more than 5 dangerous permissions, while 16.5% of the malware apps and 17% of the fraudulent apps request more than 5 permissions. Perhaps surprisingly, most legitimate

Strategy	FPR%	FNR%	Accuracy%
DT (Decision Tree)	2.46	6.03	95.98
MLP (Multi-layer Perceptron)	**1.47**	6.67	96.26
RF (Random Forest)	2.46	5.40	96.26

Table 4.3: Review classification results (10-fold cross-validation), of gold standard fraudulent (positive) and genuine (negative) reviews. MLP achieves the lowest false positive rate (FPR) of 1.47%.

(69%), malware (76%) and fraudulent apps (61%) request between 1 and 5 dangerous permissions.

After a recent Google Play policy change [Per14], Google Play organizes app permissions into groups of related permissions. Apps can request a group of permissions and gain implicit access also to dangerous permissions. Upon manual inspection of several apps, we identified a new type of malicious intent possibly perpetrated by deceptive app developers: apps that seek to attract users with minimal permissions, but later request dangerous permissions. The user may be unwilling to uninstall the app "just" to reject a few new permissions. We call these *Jekyll-Hyde apps*. Figure 4.9(c) shows the dangerous permissions added during different version updates of one gold standard malware app.

JH features. We extract the following features (see Table 4.1), (i) the total number of permissions requested by the app, (ii) its number of dangerous permissions, (iii) the app's number of dangerous permission ramps, and (iv) its total number of dangerous permissions added over all the ramps.

4.5 Evaluation

4.5.1 Experiment Setup

We have implemented FairPlay using Python to extract data from parsed pages and compute the features, and the R tool to classify reviews and apps. We have set the threshold density value θ to 3, to detect even the smaller pseudo cliques.

We have used the Weka data mining suite [Wek] to perform the experiments, with default settings. We experimented with multiple supervised learning algorithms. Due to space constraints, we report results for the best performers: MultiLayer Perceptron (MLP) [Gal90], Decision Trees (DT) (C4.5) and Random Forest (RF) [Bre01], using 10-fold cross-validation [Koh95]. For the backpropagation algorithm of the MLP classifier, we set the learning rate to 0.3 and the momentum rate to 0.2. We used MySQL to store collected data and features. We use the term "positive" to denote a fraudulent review, fraudulent or malware app; FPR means *false positive rate*. Similarly, "negative" denotes a genuine review or benign app; FNR means *false negative rate*.

We use the Receiver Operating Characteristic (ROC) curve to visually display the trade-off between the FPR and the FNR. TPR is the true positive rate. The Equal Error Rate (EER) is the rate at which both positive and negative errors are equal. A lower EER denotes a more accurate solution.

4.5.2 Review Classification

To evaluate the accuracy of FairPlay's fraudulent review detection component (RF module), we used the gold standard datasets of fraudulent and genuine reviews of § 4.3.2. We used GPCrawler to collect the data of the writers of these reviews, including the 203 reviewers of the 406 fraudulent reviews (21,972 reviews for 2,284 apps) and the 315 reviewers of the genuine reviews (9,468 reviews for 7,116 apps).

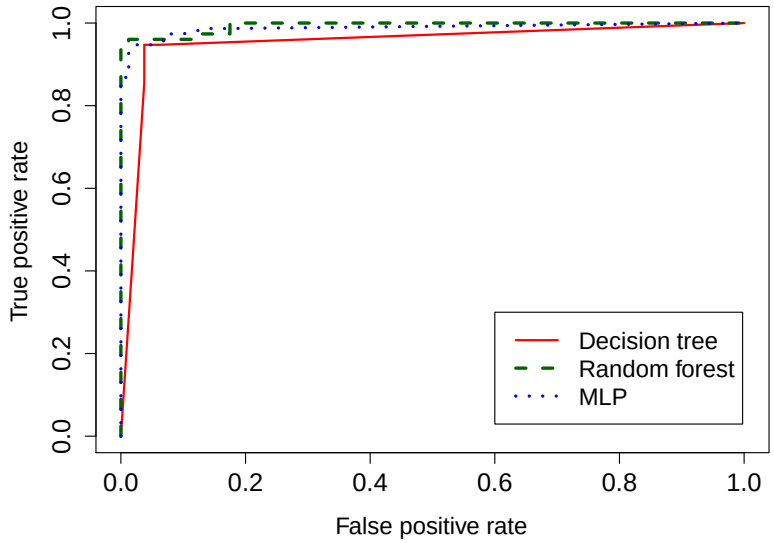


Figure 4.13: ROC plot of 3 classifiers: Decision Tree, Random Forest and Multilayer Perceptron (MLP). for review classification. RF and MLP are tied for best accuracy, of 96.26%. The EER of MLP is as low as 0.036.

We observe that the users who post genuine reviews write fewer reviews in total than those who post fraudulent reviews; however, overall, those users review more apps in total. We have also collected information about each of these collected apps, e.g., the identifiers of the app developer.

Table 4.3 shows the results of the 10-fold cross validation of algorithms classifying reviews as genuine or fraudulent (Random Forest, Decision Tree and MLP). Figure 4.13 shows the ROC plots of these algorithms. To minimize wrongful accusations, we seek to minimize the FPR [CNW+11]. MLP simultaneously achieves the highest accuracy of 96.26% and the lowest FPR of 1.47% (at 6.67% FNR). The EER of MLP is 3.6% and its area under the curve, AUC, is 0.98. Thus, in the following experiments, we use MLP to filter out fraudulent reviews in the RF.1 step.

Strategy	FPR%	FNR%	Accuracy%
FairPlay/DT	3.01	3.01	96.98
FairPlay/MLP	1.51	3.01	97.74
FairPlay/RF	**1.01**	3.52	97.74

Table 4.4: FairPlay classification results (10-fold cross validation) of gold standard fraudulent (positive) and benign apps. RF has lowest FPR, thus desirable [CNW+11].

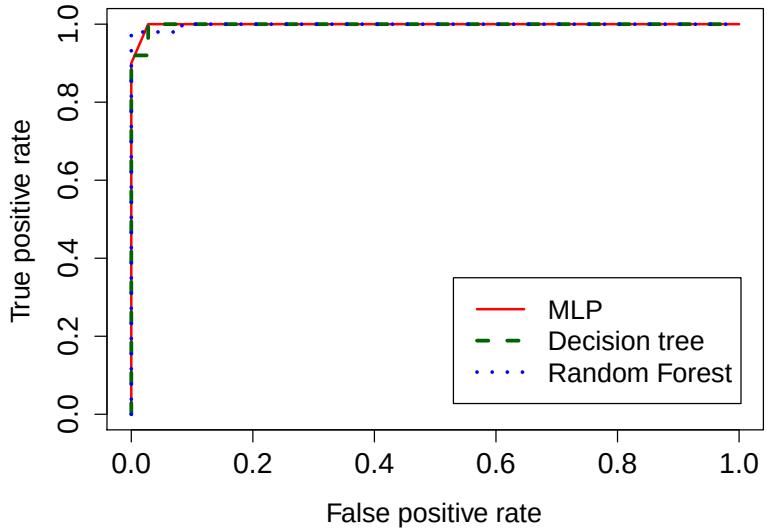


Figure 4.14: ROC plot of 3 classifiers: Decision Tree, MLP and Bagging for app classification (legitimate vs fraudulent). Decision Tree has the highest accuracy, of 98.99%. The EER of MLP is as low as 0.01.

4.5.3 App Classification

To evaluate FairPlay, we have collected all the 97,071 reviews of the 613 gold standard malware, fraudulent and benign apps, written by 75,949 users, as well as the 890,139 apps rated by these users.

In the following, we evaluate the ability of various supervised learning algorithms to correctly classify apps as either benign, fraudulent or malware. Specifically, in the first experiment we train only on fraudulent and benign app data, and test the ability to accurately classify an app as either fraudulent or benign. In the second experiment,

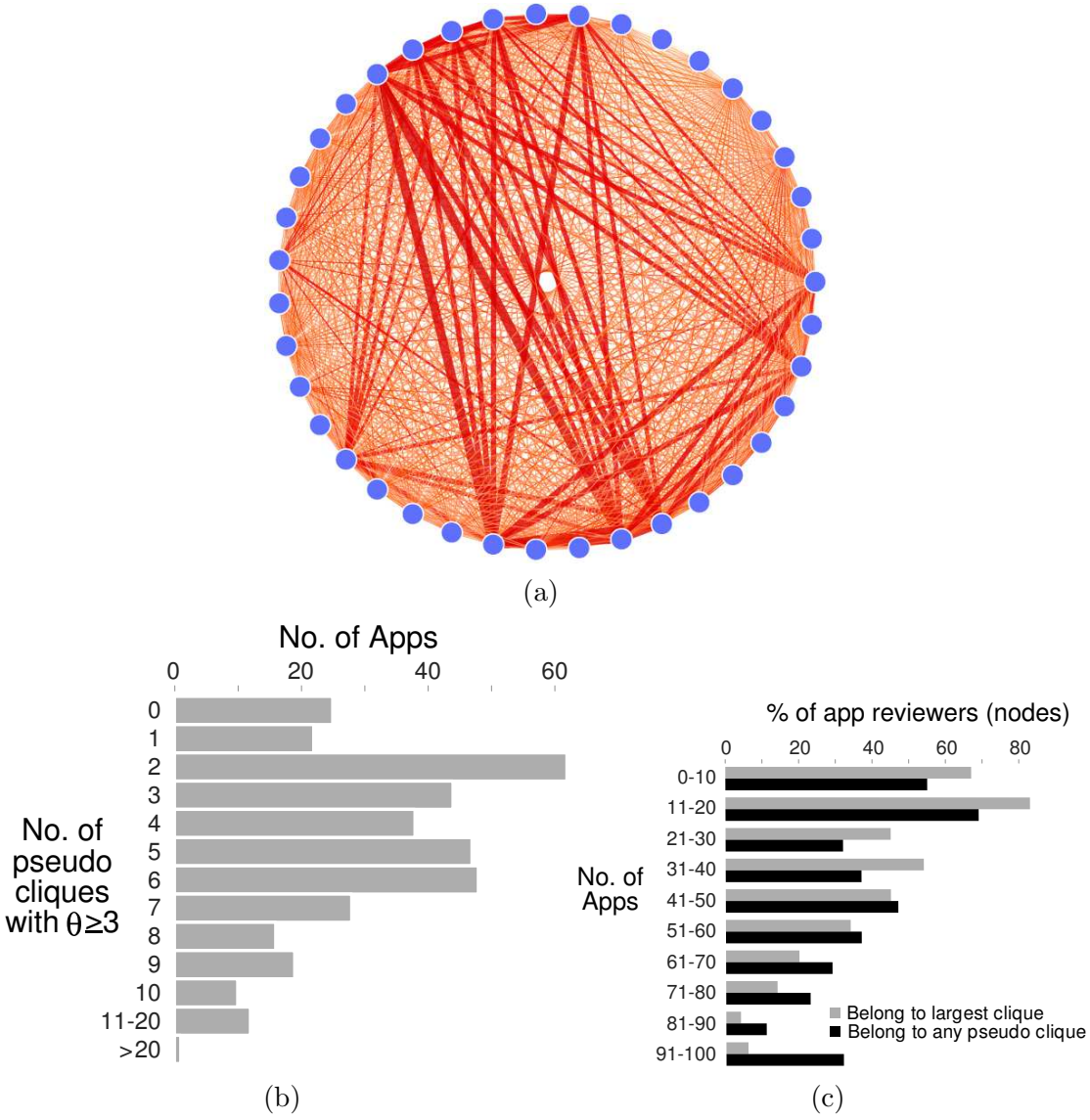


Figure 4.15: (a) Clique flagged by PCF for Tiempo - Clima gratis, one of the 201 seed fraud apps (see § 4.3.2). The clique contains 37 accounts (names hidden for privacy) that reviewed the app. The edge weights are suspiciously high: any two of the 37 accounts reviewed at least 115 apps and up to 164 apps in common! (b & c) Statistics over the 372 fraudulent apps out of 1,600 investigated: (b) Distribution of per app number of discovered pseudo-cliques. 93.3% of the 372 apps have at least 1 pseudo-clique of $\theta \geq 3$ (c) Distribution of percentage of app reviewers (nodes) that belong to the largest pseudo-clique and to any clique. 8% of the 372 apps have more than 90% of their reviewers involved in a clique!

Strategy	FPR%	FNR%	Accuracy%
FairPlay/DT	4.02	4.25	95.86
FairPlay/MLP	4.52	4.72	95.37
FairPlay/RF	**1.51**	6.13	**96.11**
Sarma et al. [SLG+12]/SVM	65.32	24.47	55.23

Table 4.5: FairPlay classification results (10-fold cross validation) of gold standard malware (positive) and benign apps, significantly outperform Sarma et al. [SLG+12]. FairPlay's RF achieves 96.11% accuracy at 1.51% FPR.

we train and test only on malware and benign apps. In the third experiment, we train a classifier on fraudulent and benign apps, then test its accuracy to classify apps as either malware or benign. Finally, we study the most impactful features when classifying fraudulent vs. benign and malware vs. benign apps.

We seek to identify the algorithms that achieve low FPR values, while having a reasonable FNR [CNW+11, TRC14]. The reason for this is that incorrectly labeling a benign app (e.g., Facebook's client) as fraudulent or malware can have a disastrous effect.

Fraud Detection Accuracy. Table 4.4 shows 10-fold cross validation results of FairPlay on the gold standard fraudulent and benign apps (see § 4.3.2). All classifiers achieve an accuracy of around 97%. Random Forest is the best, having the highest accuracy of 97.74% and the lowest FPR of 1.01%. Its EER is 2.5% and the area under the ROC curve (AUC) is 0.993 (see Figure 4.14).

Figure 4.15(a) shows the co-review subgraph for one of the seed fraud apps identified by FairPlay's PCF. The 37 accounts that reviewed the app form a suspicious tightly connected clique: any two of those accounts have reviewed at least 115 and at most 164 apps in common.

Malware Detection Accuracy. We have used Sarma et al. [SLG+12]'s solution as a baseline to evaluate the ability of FairPlay to accurately detect malware. We computed Sarma et al. [SLG+12]'s RCP and RPCP indicators using the longitudi-

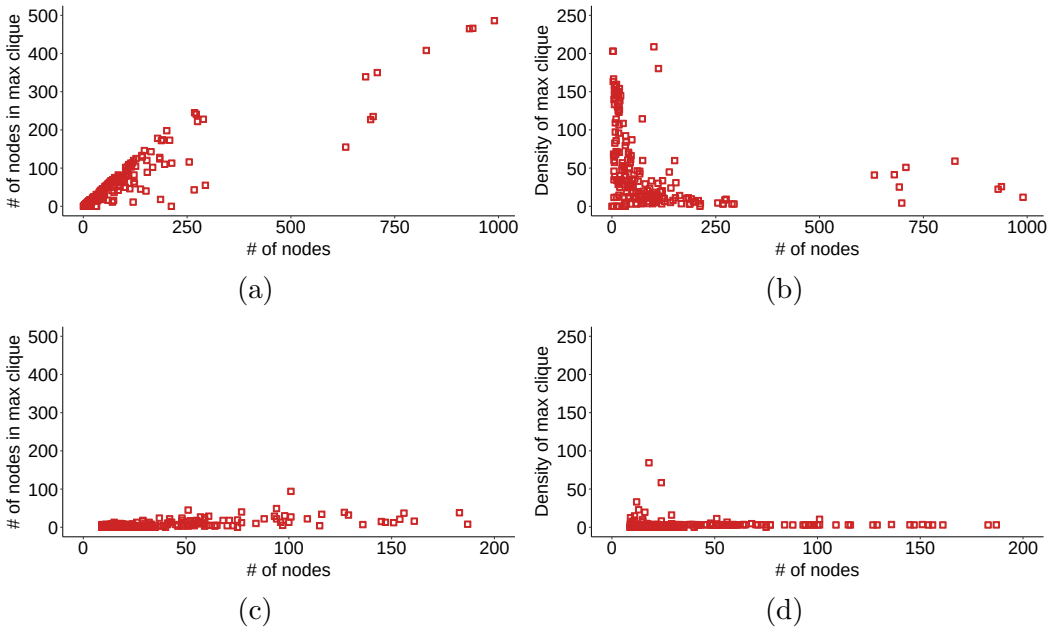


Figure 4.16: Scatterplots for the gold standard fraudulent and malware apps. (a) Each red square represents a fraudulent app, whose y axis value is its number of nodes (reviews) in the largest pseudo-clique identified, and whose x axis value is its number of nodes. (b) For each fraudulent app, the density of its largest pseudo-clique vs. its number of nodes. (c) For each malware app, the size of its largest pseudo-clique vs. its number of nodes. (d) For each malware app, the density of its largest pseudo-clique vs. its number of nodes. Fraudulent apps tend to have more reviews. While some malware apps have relatively large (but loosely connected) pseudo-cliques, their size and density is significantly smaller than those of fraudulent apps.

nal app dataset. We used the SVM based variant of Sarma et al. [SLG+12], which performs best. Table 4.4 shows 10-fold cross validation results over the malware and benign gold standard sets. FairPlay significantly outperforms Sarma et al. [SLG+12]'s solution, with an accuracy that consistently exceeds 95%. We note that the performance of Sarma et al.'s solution is lower than the one reported in [SLG+12]. This inconsistency may stem from the small number of malware apps that were used both in [SLG+12] (121 apps) and in this paper (212 apps).

For FairPlay, Random Forest has the smallest FPR of 1.51% and the highest accuracy of 96.11%. It also achieves an EER of 4% and has an AUC of 0.986. This is surprising: most FairPlay features are meant to identify search rank fraud, yet they *also* accurately identify malware.

Is Malware Involved in Fraud? We conjectured that the above result is due in part to malware apps being involved in search rank fraud. To verify this, we have trained FairPlay on the gold standard benign and fraudulent app datasets, then we have tested it on the gold standard malware dataset. MLP is the most conservative algorithm, discovering 60.85% of malware as fraud participants. Random Forest discovers 72.15%, and Decision Tree flags 75.94% of the malware as fraudulent. This result confirms our conjecture and shows that search rank fraud detection can be an important addition to mobile malware detection efforts.

Top-most Impactful Features. We further seek to compare the efficacy of FairPlay's features in detections fraudulent apps and malware. Table 4.6 shows the most impactful features of FairPlay when using the Decision Tree algorithm to classify fraudulent vs. benign and malware vs. benign apps. It shows that several features are common : the standard deviation, median and maximum over the sizes of identified pseudo-cliques (CS_{SD}, CS_{med}, CS_{max}), the number of reviews with fraud indicator words ($fraudW$). Surprisingly, even the number of reviews with malware indicator words ($malW$) has an impact in identifying fraudulent apps, yet, as expected, it has

Rank	Fraudulent vs. Benign	Malware vs. Benign
1	CS_{SD}	$nCliques$
2	$inCliqueCount$	CS_{SD}
3	$spikeCount$	CS_{med}
4	CS_{max}	$malW$
5	ρ_{max}	I_1/Rv_1
6	CS_{med}	CS_{max}
7	$fraudW$	$fraudW$
8	$malW$	$dangerCount$

Table 4.6: Top 8 most important features when classifying fraudulent vs. benign apps (center column) and malware vs. benign apps (rightmost column). Notations are described in Table 4.1. While some features are common, some are more efficient in identifying fraudulent apps than malware apps, and vice versa.

a higher rank when identifying malware apps.

In addition, as expected, features such as the percentage of nodes involved in a pseudo-clique ($inCliqueCount$), the number of days with spikes ($spikeCount$) and the maximum density of an identified pseudo-clique (ρ_{max}) are more relevant to differentiate fraudulent from benign apps. The number of pseudo-cliques with density larger than 3 ($nCliques$) the ratio of installs to reviews (I_1/Rv_1) and the number of dangerous permissions ($dangerCount$) are more effective to differentiate malware from benign apps.

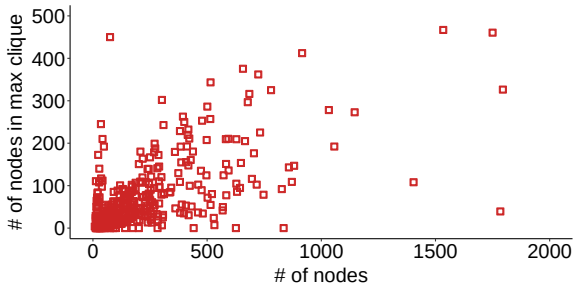
More surprising are the features that do not appear in the top, for either classifier. Most notably, the Jekyll-Hyde features that measure the ramps in the number of dangerous permissions. One explanation is that the 212 malware apps in our gold standard dataset do not have sufficient dangerous permission ramps. Also, we note that our conjecture that fraudster efforts to alter the search rank of an app will not be able to preserve a natural balance of the features that impact it (see IRR module) is only partially validated: solely the I_1/Rv_1 feature plays a part in differentiating malware from benign apps.

Furthermore, we have zoomed in into the distributions of the sizes and densities of the largest pseudo-cliques, for the gold standard fraudulent and malware apps. Figure 4.16 shows scatterplots over the gold standard fraudulent and malware apps, of the sizes and densities of their largest pseudo-cliques, as detected by FairPlay. Figure 4.16(a) shows that fraudulent apps tend to have very large pseudo-clique and Figure 4.16(c) shows that malware apps have significantly smaller pseudo-cliques. We observe however that malware apps have fewer reviews, and some malware apps have pseudo-cliques that contain almost all their nodes. Since the maximum, median and standard deviation of the pseudo-clique sizes are computed over values normalized by the app's number of reviews, they are impactful in differentiating malware from benign apps.

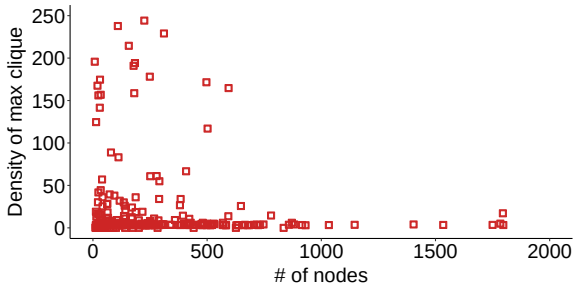
Figure 4.16(b) shows that the largest pseudo-cliques of the larger fraudulent apps tend to have smaller densities. Figure 4.16(d) shows a similar but worse trend for malware apps, where with a few exceptions, the largest pseudo-cliques of the malware apps have very small densities.

4.5.4 FairPlay on the Field

We have also evaluated FairPlay on other, non "gold standard" apps. For this, we have first selected 8 app categories: Arcade, Entertainment, Photography, Simulation, Racing, Sports, Lifestyle, Casual. We have then selected the $6,300$ apps from the longitudinal dataset of the 87K apps, that belong to one of these 8 categories, and that have more than 10 reviews. From these $6,300$ apps, we randomly selected 200 apps per category, for a total of $1,600$ apps. We have then collected the data of all their $50,643$ reviewers (not unique) including the ids of all the $166,407$ apps they reviewed.



(a)



(b)

Figure 4.17: Scatterplots of the 372 fraudulent apps out of 1,600 investigated, showing, for each app, (a) the number of nodes (reviews) in the largest clique identified vs. the app's number of nodes (b) the density of the largest clique vs. the app's number of nodes. While apps with more nodes also tend to have larger cliques, those cliques tend to have lower densities.

We trained FairPlay with Random Forest (best performing on previous experiments) on all the gold standard benign and fraudulent apps. We have then run FairPlay on the 1,600 apps, and identified 372 apps (23%) as fraudulent. The Racing and Arcade categories have the highest fraud densities: 34% and 36% of their apps were flagged as fraudulent.

Intuition. We now focus on some of the top most impactful FairPlay features to offer an intuition for the surprisingly high fraud percentage (23% of 1,600 apps). Figure 4.15(b) shows that 93.3% of the 372 apps have at least 1 pseudo-clique of $\theta \geq 3$, nearly 71% have at least 3 pseudo-cliques, and a single app can have up to 23 pseudo-cliques. Figure 4.15(c) shows that the pseudo-cliques are large and encompass many of the reviews of the apps: 55% of the 372 apps have at least 33% of their reviewers involved in a pseudo-clique, while nearly 51% of the apps have a

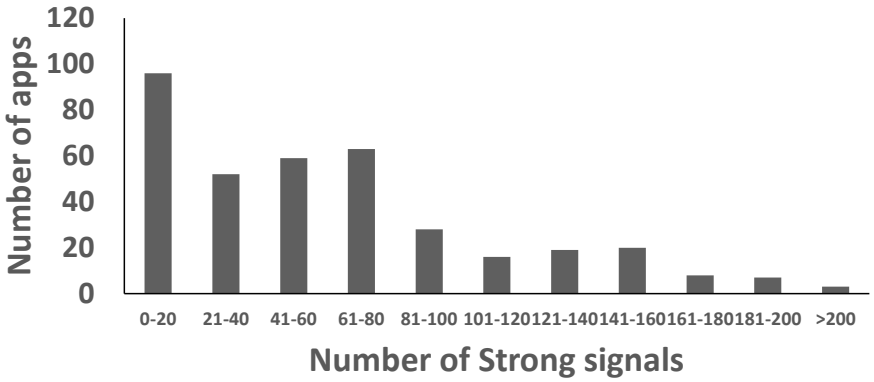


Figure 4.18: Distribution of the number of malware and fraud indicator words (see Step RF.2) in the reviews of the 372 identified fraudulent apps (out of 1,600 apps). Around 75% of these apps have at least 20 fraud indicator words in their reviews.

single pseudo-clique containing 33% of their reviewers.

Figure 4.17 shows the scatterplots of the number of nodes and densities of the largest clique in each of the 372 apps. While intuitively apps with more reviews tend to have larger pseudo-cliques (Figure 4.17(a)), surprisingly, the densities of such pseudo-cliques are small (Figure 4.17(b)).

Figure 4.18 shows the distribution of the number of malware and fraud indicator words (see Step RF.2) in the reviews of the identified 372 fraudulent apps. It shows that around 75% of the 372 fraudulent apps have at least 20 fraud indicator words in their reviews.

4.5.5 Coercive Review Campaigns

Upon close inspection of apps flagged as fraudulent by FairPlay, we detected apps perpetrating a new attack type: harass the user to either (i) write a positive review for the app, or (ii) install and write a positive review for other apps (often of the same developer). We call these behaviors *coercive review campaigns* and the resulting reviews, as *coerced reviews*. Example coerced reviews include, "I only rated it because

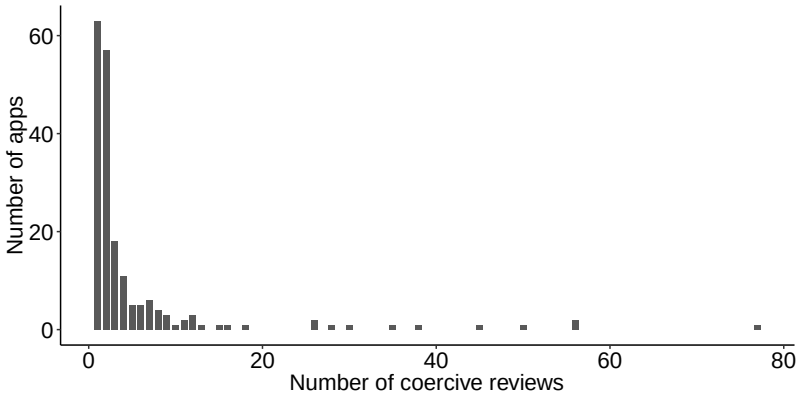


Figure 4.19: Distribution of the number of coerced reviews received by the 193 coercive apps we uncovered. 5 apps have each received more than 40 reviews indicative of rating coercion, with one app having close to 80 such reviews!

i didn't want it to pop up while i am playing", or "Could not even play one level before i had to rate it [...] they actually are telling me to rate the app 5 stars".

In order to find evidence of systematic coercive review campaigns, we have parsed the 2.9 million reviews of our dataset to identify those whose text contains one of the root words ["make", "ask", "force"] and "rate". Upon manual inspection of the results, we have found $1,024$ coerced reviews. The reviews reveal that apps involved in coercive review campaigns either have bugs (e.g., they ask the user to rate 5 stars even after the user has rated them), or reward the user by removing ads, providing more features, unlocking the next game level, boosting the user's game level or awarding game points.

The $1,024$ coerced reviews were posted for 193 apps. Figure 4.19 shows the distribution of the number of coerced reviews per app. While most of the 193 apps have received less than 20 coerced reviews, 5 apps have each received more than 40 such reviews.

We have observed several duplicates among the coerced reviews. We identify two possible explanations. First, as we previously mentioned, some apps do not keep track of the user having reviewed them, thus repeatedly coerce subsequent reviews

from the same user. A second explanation is that seemingly coerced reviews, can also be posted as part of a negative search rank fraud campaign. However, both scenarios describe apps likely to have been subjected to fraudulent behaviors.

4.6 Research Contributions Acknowledgment

It is a pleasure to acknowledge the contributions of my co-authors; Dr. Mahmudur Rahman, Dr. Duen Horng Chau and my supervisor Dr. Bogdan Carbunar for their roles in the teamwork. Dr. Mahmudur Rahman and me, both carried out the experiments together and derived the models and analyzed the data. My special gratitude goes to Dr. Carbunar and Dr. Chau for providing critical feedback to shape the research, analysis, and manuscript.

4.7 Summary

We have introduced FairPlay, a system to detect both fraudulent and malware Google Play apps. Our experiments on a newly contributed longitudinal app dataset, have shown that a high percentage of malware is involved in search rank fraud; both are accurately identified by FairPlay. In addition, we showed FairPlay's ability to discover hundreds of apps that evade Google Play's detection technology, including a new type of coercive fraud attack.

CHAPTER 5

Real Time Online Fraud Preemption

5.1 Motivation

The social impact of online services built on information posted by their users has also turned them into a lucrative medium for fraudulently influencing public opinion [RRCC16b, BSLL+16b, LZ16, SWE+13]. The need to aggressively promote disinformation has created a black market for social network fraud, that includes fake opinions and reviews, likes, followers and app installs [MML+11, TSM16, RR16, RL16, AV16, AS16, AR16]. For instance, in § 5.3.1, we show that in fraud markets, a fake review can cost between \$0.5 and \$3 and a fake social networking "like" can cost \$2. The profitability of fraud suggests that current solutions that focus on fraud detection, are unable to control organized fraud.

In this paper we introduce the concept of *fraud preemption systems*, solutions deployed to defend online systems such as social networks and app markets. Instead of reacting to fraud posted in the past, fraud preemption systems seek to discourage fraudsters from posting fraud in the first place. We propose FraudSys, the first real-time fraud preemption system that reduces the profitability of fraud from the perspective of both crowdsourced fraud workers and the people who hire them. FraudSys imposes computational penalties: the activity of a user (e.g., review, like) is posted online only after his device solves a computational puzzle. Puzzles reduce the profitability of fraud by (i) limiting the amount of fraud per time unit that can be posted for any subject hosted on the online system, and (ii) by consuming the computational resources of fraudsters. For instance, Figure 5.1 shows the timelines of daily penalties assigned by FraudSys to two fraudsters detected in Google Play. Based only on the recorded activities, FraudSys frequently assigned hundreds of hours of daily computational penalties to a single fraudster.

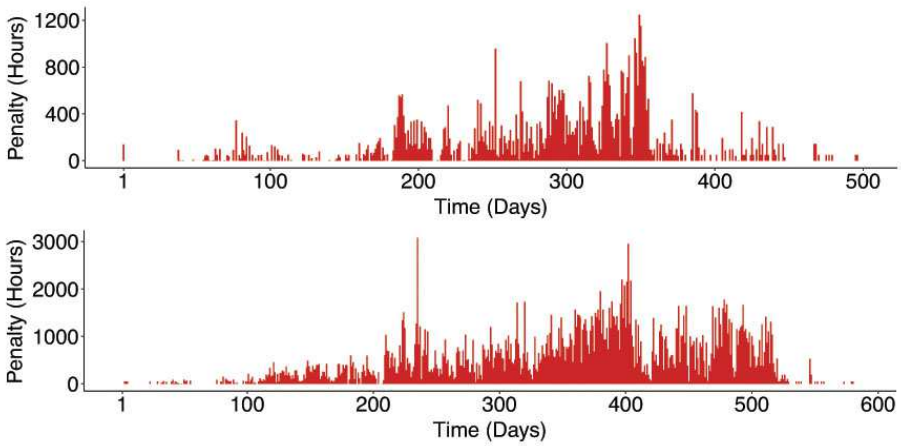


Figure 5.1: Timeline of daily penalties (in hours) assigned by FraudSys to the Google Play activities of two fraudsters we identified in Freelancer.com. FraudSys imposes daily penalties of up to 1,247 hours to the fraudster at the top and 3,079 hours for the fraudster at the bottom. As a result, the fraudsters need to consume significant computational resources, while their fraud is significantly delayed. This in turn reduces the number of payments they would receive, and impacts their profitability.

Challenges. Implementing a fraud preemption system raises several challenges. First, FraudSys needs to detect fraud in real-time, whenever a user performs an online system activity. Once assigned, a puzzle cannot be rescinded. This is in contrast to existing systems (e.g., Yelp) that detect fraud retroactively and can update previous decisions when new information surfaces. Second, FraudSys needs to impose difficult puzzles on fraudsters, but minimally impact the experience of honest users. This is made even more complex by the fact that fraudsters can attempt to bypass detection and even obscure their true ability to solve puzzles. Third, a stateful FraudSys service that maintains state for millions of issued and active puzzles is expensive and vulnerable to DoS attacks.

Our Contributions. Through FraudSys, we introduce several innovative solutions. To address the first challenge, we exploit observations of fraudulent behaviors gleaned from crowdsourcing sites and online systems, to propose a real-time graph based algorithm to infer an *activity fraud score*, the chance that a user activity is fraudulent

[§ 5.4.2]. More specifically, we introduce features that group fraudulent activities according to their human creator: FraudSys identifies densely connected components in the co-review graph of the subject targeted by the user activity, each presumably controlled by a different fraudster. It then quantifies the connectivity of the user account performing the action, to each component, and uses the highest connectivity as features that may indicate that the user account and the corresponding component are controlled by the same fraudster. FraudSys then leverages supervised learning algorithms trained on these features to infer the activity fraud score.

To address the second challenge, we develop adaptive hashrate inference techniques to detect the computational capabilities of even adversarial controlled devices to solve puzzles [§ 5.4.3], and devise mechanisms to convert fraud scores to appropriate temporal penalty and puzzle difficulty values [§ 5.4.3]. The puzzles assigned by FraudSys do not alter the online experience of users, as they are solved on their devices, in the background. However, the puzzles (1) significantly delay detected fraudulent activities, posted only when the device returns the correct puzzle solutions and (2) consume the computational resources of the fraudsters who control the devices.

To address the third challenge, we propose the notion of *stateless computational puzzles*, computational tasks that impose no storage overhead on the fraud preemption system provider, but enable it to efficiently verify their authenticity and the correctness of their solutions [§ 5.4.1]. Thus, the fraud preemption system can assign a puzzle to a device from which an activity was performed on the online system, without storing any state about this task. The device can return the results of the puzzle in 5 seconds or 1 day, and the provider can verify that the task is authentic, and its results are correct. This makes our approach resistant to DoS attacks that attempt to exhaust the provider's storage space for assigned puzzles.

We show that the computational penalty imposed by FraudSys on a fraudulent activity is a function of the capabilities of the device from which it is performed, and the probability that the activity is fraudulent. We introduce and prove upper bounds on the profitability of fraud and the amount of fraud that can be created for a single subject, per time unit [§ 5.5] . We evaluate FraudSys on 23,028 fraudulent reviews (posted by 23 fraudsters from 2,664 user accounts they control), and 1,061 honest reviews we collected from Google Play, as well as 274,297 fake and 180,400 honest likes from Facebook. Even with incomplete data, FraudSys imposes temporal penalties that can be as high as 3,079 hours per day for a single fraudster. We also show that fraud does not pay off. At today's fraud payout, a fraudster equipped with an AntMiner S7 (Bitcoin mining hardware) will earn through fraud less than half the payout of honest Bitcoin mining.

5.2 Related Work

Computation Based Fraud Preemption. Dwork and Naor [DN92] were the first to propose the use of computation to prevent fraud, in particular spam, where the sender of an e-mail needs to include the solution to a "moderately hard function" computed over a function of the e-mail. Juels and Brainard [JB99] proposed to use puzzles to prevent denial of service attacks, while Borisov [Bor06] introduced puzzles that deter Sybils in peer-to-peer networks. In Borisov [Bor06], newly joined peers need to solve a puzzle to which all the other peers have contributed.

FraudSys not only seeks to adapt computational puzzles to prevent online system fraud, but also needs to solve the additional challenges of building puzzles whose difficulty is a function of the probability that an activity is fraudulent, while handling heterogeneous user devices (e.g., ranging from smartphones to machines that specialize in such puzzles).

Graph Based Fraud Detection. Graphs have been used extensively to model relationships and detect fraudulent behaviors in online systems. Ye and Akoglu [YA15a] quantified the chance of a subject to be a spam campaign target, then clustered spammers on a 2-hop subgraph induced by the subjects with the highest chance values. Lu et al. [LZXL13] proposed a belief propagation approach implemented on a review-to-reviewer graph, that simultaneously detects fake reviews and spammers (fraudsters).

Mukherjee et al. [MLG12] proposed a suite of features to identify reviewer groups, as users who review many subjects in common but not much else, post their reviews within small time windows, and are among the first to review the subject. Hooi et al. [HSB+16b] have recently shown that fraudsters have evolved to hide their traces, by adding spurious reviews to popular items. To identify "camouflaged" fraud, Hooi et al. [HSB+16b] introduced "suspiciousness" metrics that apply to bipartite user-to-item graphs, and developed a greedy algorithm to find the subgraph with the highest suspiciousness. Akoglu et al. [ATK15] survey graph based online fraud detection. [FH16] provide a survey of community detection methods, evaluation scores and techniques for general networks.

Unlike previous work, FraudSys assigns fraud scores to individual user activities in *real time*, thus uses only partial information. To achieve this, FraudSys develops and leverages features that quantify the connectivity of the user activity to other groups of activities *previously* performed by other fraudsters on the same subject. Further, FraudSys also imposes computation and temporal penalties to discourage fraud creation.

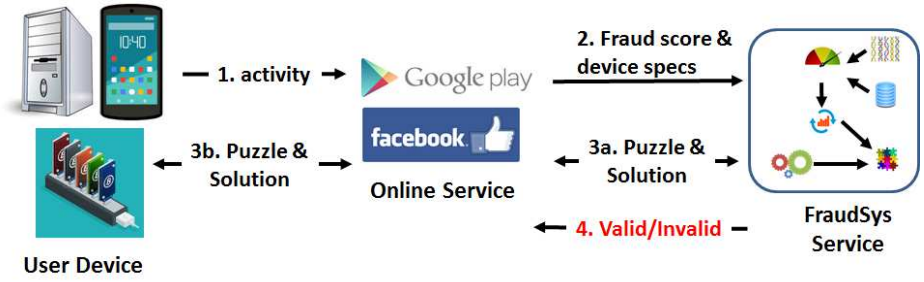


Figure 5.2: System model. The user performs actions on the online service, from a device that can range from a smartphone to a Bitcoin miner. The online service implements and posts the activity only if and after the FraudSys service validates it. The FraudSys functionality can be implemented by the online service or by a third party provider.

5.3 System and Adversary Model

Figure 5.2 illustrates the three main components of the system model. First, the online service (the *service*) hosts the system functionality, and stores information about user accounts and featured *subjects*. Subjects can be apps in stores like Google Play, or pages for businesses, accounts and stories in social networks like Facebook.

Second, the users: they register with the service, record profile information (e.g., name) and receive initial service credentials, including a unique id. Users can access the online service from a variety of devices. For this, they need to install a client (e.g., app) on each device they use. The online service stores and maintains information about each device that the user has used, e.g., to provide compatibility information on Google Play apps.

Users are encouraged to *act* on existing subjects. The activities include posting reviews, comments, or likes, installing mobile apps, etc. The online service associates statistics over the activities performed for each supported subject. The statistics have a significant impact on the popularity and search rank of subjects [Ank13, App16], thus are targets of manipulation by fraudsters (see § 5.3.1).

The third component of the system model is the FraudSys service, whose goal is to validate user activities. For increased flexibility, Figure 5.2 shows FraudSys as an independent provider. However, FraudSys can also be a component of the online service.

5.3.1 Adversary Model

We consider two types of adversaries – adversarial owners and crowdsourced fraud workers.

Adversarial owners. Adversarial behaviors start with the subject owners. Adversarial owners seek to fraudulently promote their subjects (or demote competitor subjects) in order to bias the popularity and public opinion of specific subjects. For instance, fraudulent promotions seek to make subjects more profitable [LZ16, AM12], increase the "reachability" of malware (through more app installs), and boost the impact of fake news.

Fraud workers (= fraudsters). We assume that adversarial owners crowdsource this promotion task (also known as search rank fraud) to *fraud workers*, or fraudsters. In this paper we focus on two types of fraudulent activities: writing fake reviews in Google Play and posting fake "Likes" in Facebook. We have studied fraudster recruitment jobs in crowdsourcing sites and fraud posted in Google Play and Facebook. This has allowed us to collect fraud data (see § 5.6.1) and to identify several fraud behaviors: (i) more than one fraudster can target the same subject; (ii) user accounts controlled by a fraudster tend to have a significant history of common activities, i.e., performed on the same subjects; and (iii) accounts controlled by different fraudsters tend to have few common past activities.

Fraud incentives. We assume that fraud workers are rational, motivated by financial incentives. That is, given an original investment in expertise and equipment, a fraud

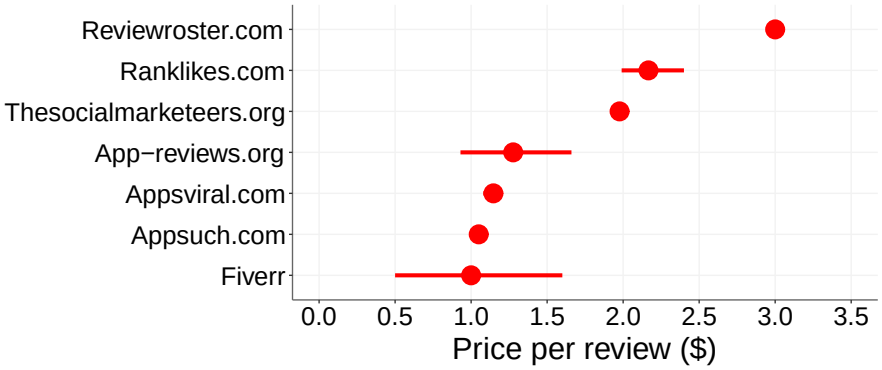


Figure 5.3: Price per review (minimum, average and maximum), for crowdsourcing sites that focus on app market fraud. The sites offer "fraud packages" and even discounts for bulk fake review purchases. A fake review costs between \$0.5-\$3.

worker seeks to maximize his revenue achieved per time unit. Figure 5.3 shows the minimum, average and maximum cost per fraudulent activity, as advertised by several crowdsourcing and fraud-as-a-service (FAAS) sites: a fake review for an app is worth between \$0.5-\$3, while a fake social networking "like" can cost \$2. In contrast, an adversarial owner may have both financial incentives (e.g., increased market share for his subject, thus revenue), and external incentives (e.g., malware or fake news distribution).

5.3.2 Fraud Preemption System Definition

We introduce the concept of *fraud preemption systems*, that seek to restrict the profitability of fraud for both fraudsters and the people who hire the fraudsters (i.e., adversarial owners). Specifically, let $Sys = (\mathcal{U}, \mathcal{S}, \mathcal{F}, P)$ be a system that consists of finite sets of users ($\mathcal{U}$), subjects ($\mathcal{S}$) and fraudsters ($\mathcal{F}$) that interact through a set of procedures P. In the adversary model of § 5.3.1, we say that Sys is a (p,a)-*fraud preemption system* if it satisfies the following two conditions:

1. **Fraudster deterrence**: The average payout per time unit of any fraudster in $\mathcal{F}$ does not exceed p.

2. **Adversarial owner deterrence**: The average number of fraudulent activities allowed for any subject in $\mathcal{S}$ per time unit does not exceed a.

In addition, a puzzle-based fraud preemption system needs to satisfy the following requirements:

1. **Real-time fraud detection**. Detect fraud at the time it is created, with access to only limited information (i.e., no knowledge of the future).
2. **Penalty accuracy**. Impose difficult puzzles on fraudsters, but minimally impact the online experience of honest users.
3. **Device heterogeneity**. Both honest and fraudulent users may register and use multiple devices to access the online service. Malicious users may obfuscate the computational capabilities of their devices.
4. **Minimize system resource consumption**. The high number of issued, active puzzles will consume the resources of the FraudSys provider, and open it to DoS attacks.

5.4 FraudSys

We introduce FraudSys, a real-time fraud preemption system that requires users to verify commitment through an imposed resource consumption action for each activity they perform on the online system. Specifically, FraudSys requires the device from which the activity was issued, to solve a *computational puzzle*. FraudSys consists of the modules illustrated in Figure 5.4: The Fraud Detection module takes as input a user activity and the current state of the subject, and outputs a *fraud score*. The Fraud2Penalty module converts the fraud score to a *time penalty*: the time that the user's device will need to spend working on a computational puzzle. The Hashrate Inference module interacts with the user device in order to learn its puzzle solving

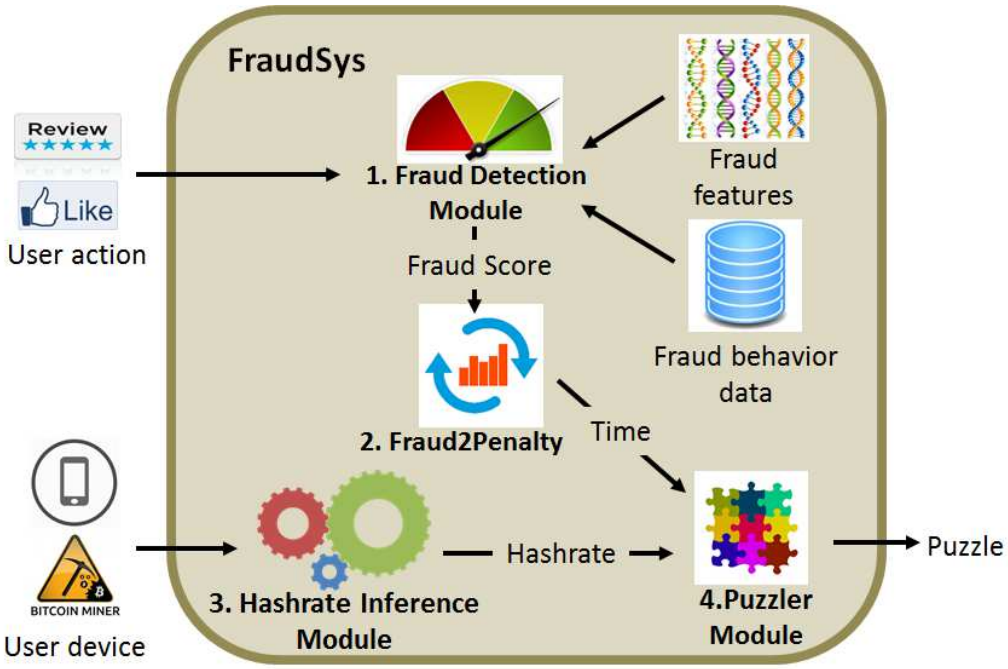


Figure 5.4: FraudSys architecture. The Fraud Detector module uses supervised learning to assign a fraud score to user activities. The Fraud2Penalty module converts the fraud score to a time penalty. The Hashrate Inference module estimates the computational capabilities of the user device. Finally, the Puzzler module generates a puzzle that the device should take approximately the time penalty to solve.

capabilities. Finally, the Puzzler module uses the inferred device capabilities to generate a puzzle that the device will take a time approximately equal to *time penalty* to solve.

To address requirement #1, the Fraud Detection module exploits the fraudulent behaviors described in § 5.3.1. It builds *co-activity graphs* and extracts features that model the relationships between the user performing the activity and other users that have earlier performed similar activities for the same subject.

We address requirement #2 through a two-pronged approach. First, the Fraud Detection and Fraud2Penalty modules ensure that the difficulty of a FraudSys puzzle will be a function of the detected probability of fraud: activities believed to be honest will be assigned trivial puzzles, while increasingly fraudulent activities will be assigned increasingly difficult puzzles. Second, FraudSys does not change the experience of the user on the online system: the user writes the review or clicks on the like button, then continues browsing or quits the app. The assigned puzzle is solved in the background by the device on which the activity was performed. However, FraudSys delays the publication of the activity, until the device produces the correct puzzle solution.

To address requirement #3, the Hashrate Inference module estimates the hashrate of the device performing the activity, and provides the tool to punish devices that cheat about their puzzle solving capabilities. To solve requirement #4, the Puzzler module generates puzzles that outsource the storage constraints from the FraudSys service to the user devices that solve the puzzles. In the following we detail each FraudSys module, starting with the central puzzle creation module.

5.4.1 The Puzzler Module: Stateless Puzzles

Let U be a user that performs an activity A from a device D, on a subject S hosted by the online service. Table 5.1 summarizes the notations we use. The FraudSys

Notation	Definition
U, D, S, A	user, device, subject, activity
T	time of puzzle issue
r	activity fraud score
Δ	puzzle difficulty
η_D	hashrate of device D
Γ	puzzle cookie
Π	puzzle
$target$	puzzle target value
τ	temporal penalty
q	number of shares (puzzle solutions)
K	secret key of FraudSys

Table 5.1: FraudSys symbol table.

service stores minimal state for each registered user, and *serializes* his activities, see Figure 5.5: the devices from which a user performs a sequence of activities on the online service, are assigned one puzzle per activity, each with its own timeout. The device needs to return the puzzle solutions before the associated timeout. To implement this, for each user U, the FraudSys service stores the following entry:

$$U, [\langle D_i, \eta_i \rangle]_{i=1..d}, timeout,$$

where, for each of the $i = 1..d$ devices registered by U, D_i is the device identifier and η_i is its hashrate (puzzle solving capabilities measure, see following), and $timeout$ is the latest time by which one of these devices needs to return puzzle solutions.

FraudSys builds on the computational puzzles of Bitcoin, see [Nak08]. Let $H^2(M)$ denote the double SHA-256 hash of a message M. Then, the FraudSys puzzle issued to device D consists of a $target$ value and a fixed string F. We detail F shortly. To solve the puzzle, D needs to randomly choose 32 byte long *nonce* values until it finds at least one that satisfies:

$$H^2(nonce||F) < target \tag{5.1}$$

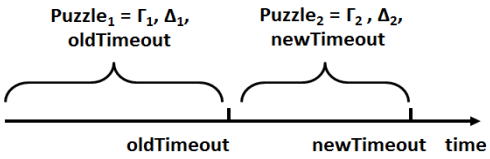


Figure 5.5: Puzzle serialization: a user can perform multiple activities, but each receives a different puzzle with its own timeout, authenticated through the cookie Γ.

That is, the double hash of the *nonce* concatenated with F, needs to be smaller than the *target* value, another 32 byte long value. A smaller *target* implies a harder puzzle. The largest *target* acceptable by the system is called $target\_1$, or *target of difficulty 1*.

Bitcoin has two drawbacks. First, the current difficulty of Bitcoin puzzles requires computational capabilities that greatly exceed those of devices used to access online services. Second, Bitcoin requires the network to maintain state about issued puzzles. State storage exposes FraudSys to attacks, while not storing state can enable adversaries to lower the difficulty of their assigned puzzles. To address these problems we (i) change the $target\_1$ difficulty to allow trivial puzzles, and (ii) introduce *puzzle cookies*, special values that authenticate puzzles with minimal FraudSys state, see following.

Device hashrate and puzzle difficulty. We set the $target\_1$ value to be a 32 byte long value with one zero at the beginning, e.g., $2^{255}-1$. In addition, the *hashrate* η_D of a device D is a measure that describes the ability of the device to solve puzzles. Since the puzzles need to be solved in a brute force approach, the hashrate is measured in hashes per second. A relevant concept is the notion of *difficulty*, denoted by Δ, a measure of how difficult it is to solve a puzzle whose input values hash below a given target. Its relationship to the above *target* value is given by:

$$\Delta = \frac{target\_1}{target} = \frac{2^{255}-1}{target} \tag{5.2}$$

Given η_D, we derive the time τ taken by D to solve a puzzle with difficulty Δ, as

follows. First, the number of hashes smaller than a given target is equal to the target. For instance, the number of hashes smaller than $target\_1$ is $2^{255} - 1$. Then, the probability p of finding an input that hashes to a value smaller than the target is equal to the target divided by the total number of hashes (2^{256}). Furthermore, the expected number of hashes, E, before achieving the target is given by $1/p$. Thus:

$$E = \eta_D \times \tau = \frac{2^{256}}{target} = \frac{2^{256}}{target\_1} \times \frac{target\_1}{target} \approx 2 \times \Delta$$

and conclude that

$$\tau = \frac{2 \times \Delta}{\eta_D} \tag{5.3}$$

For instance, the lowest puzzle difficulty is 1, which occurs when the $target$ has a prefix of one zero and the device is expected to generate 2 hashes before solving the puzzle. Similarly, the maximum difficulty is $(2^{255} - 1)$, for a $target = 1$, when the device is expected to perform $\frac{2^{255}-1}{1} \times 2 \approx 2^{256}$ hashes.

The FraudSys puzzle and cookies. To minimize the storage imposed on the FraudSys service (see above), we leverage the cookie concept [Ber96]. Algorithm 2 illustrates the puzzle creation, verification and computation components. The FraudSys service generates and stores a secret key K (line 2). When a user U performs an activity A from a device D on a subject S of the online service, the online service calls the BuildCookie function of the FraudSys service (lines 3-11).

Algorithm 2 FraudSys puzzle creation, verification and computation components.

```
1. Object FraudSysService
2.    K: key;
3.    Function BuildCookie(U, D, S, A, q)
4.       η_D := getHashrate(U, D);
5.       r := computeFraudScore(U, S, A);
6.       τ := fraud2Penalty(r);
7.       Δ := η_D × τ/2q
8.       oldT := getTimeout(U);
9.       newT := oldT + τ;
10.      Γ := HMAC(K, U, D, S, newT, Δ, A);
11.      setTimeout(U, newT);
12.      return Γ, Δ, newT;
13.   Function VerifyPuzzle(U, D, S, A, timeout, Γ, σ: share[q])
14.      if (Γ != HMAC(K, U, D, S, A, timeout, Δ) return -1;
15.      target := getTarget(Δ);
16.      for (i := 0; i < q; i++)
17.         if (H^2(σ[i] || Γ) > target) return -1;
18.      waitUntil(timeout); post A;
19.      τ' := T_c − T;
20.      if ((η_D := 2Δ/τ') ≥ η_min)
21.         updateHashrate(U, D, η_D);
22. Object UserDevice
23.   Function SolvePuzzle(Γ, Δ, timeout, q)
24.      target := getTarget(Δ);
25.      σ := new share[q]; i := 0;
26.      while (i < q) do
27.         nonce := getRandom();
28.         if (H^2(nonce || Γ) < target)
29.            σ[i] := nonce;
30.            i := i+1;
31.      return U, D, S, A, timeout, Γ, σ;
```

BuildCookie retrieves the hashrate of the device D from the record stored by FraudSys for U (line 4). It then computes the fraud score associated to the activity (line 5) then converts it to a time penalty τ (line 6). We describe this functionality in the next subsections. BuildCookie then uses a modified Equation 5.3 to compute the difficulty Δ that the puzzle should have (line 7). Δ is q times smaller than in Equation 5.3, as the puzzle solution consists of q shares, see SolvePuzzle.

BuildCookie gets the current timeout $oldT$ of U, and updates it to $newT$ by adding the penalty τ to it (lines 8-9). It then computes the puzzle cookie Γ,

$$\Gamma = HMAC_K(U, D, S, A, timeout, \Delta)$$

as a keyed HMAC [BCK96] over the user and device id, subject, activity, new timeout and puzzle difficulty (lines 9-10). BuildCookie sets U's timeout value to the updated $newT$ value (line 11), then returns the following puzzle (line 12) to the online service that forwards it to device D (see Figure 5.2):

$$\Pi = \Gamma, \Delta, timeout.$$

The puzzle cookie ensures that an adversary that modifies the puzzle's difficulty or timeout, will be detected: the adversary does not know the key K, which is a secret of the FraudSys service. Puzzle cookies are unique with high probability, due to collision resistance properties of the HMAC, whose input is non-repeating.

Solving the puzzle. When the device D receives the puzzle, it needs to solve it: search for q *nonce* values that satisfy the inequality $H^2(nonce \mid\mid \Gamma) < target$, for a $target$ corresponding to the difficulty Δ. Specifically, D invokes the SolvePuzzle function (lines 23-31), that needs to identify q *shares*, i.e., nonce values that satisfy the puzzle. q is a system parameter. The function first uses Equation 5.2 to retrieve the $target$ value corresponding to the difficulty Δ (line 24). Then, it generates random *nonce* values until it identifies q values that satisfy the puzzle condition (lines 25-30). SolvePuzzle returns the identified shares (in the σ array), which are then sent to the online service and forwarded to the FraudSys server, along with the user, device and subject ids, activity, timeout and cookie of the received puzzle (see line 12 and Figure 5.2).

Verification of puzzle correctness. Upon receiving these values, the FraudSys server invokes the VerifyPuzzle function (lines 13-21), to verify its correctness as

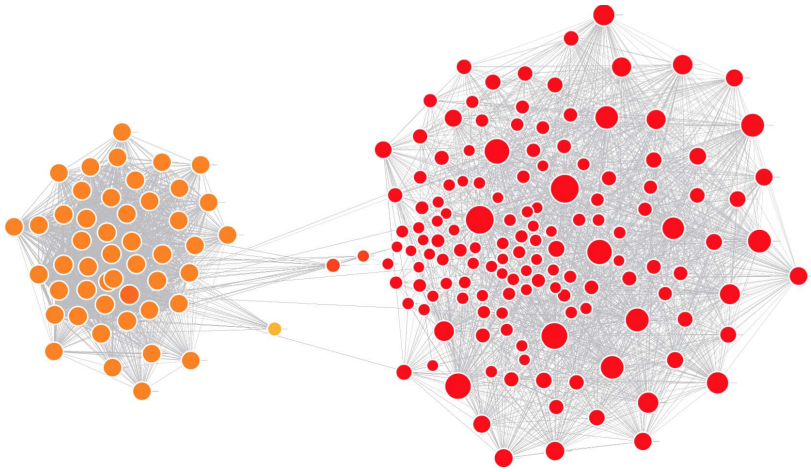


Figure 5.6: Visualization of the co-review graph of a fraudulent Google Play app. The nodes represent user accounts; edges connect nodes corresponding to accounts with common, past review activities. The nodes in each of the 2 clusters correspond to accounts controlled by the same fraudster.

follows: (1) Reconstruct the puzzle cookie Γ based on the received values and the secret key K. Verify that this cookie is equal to the received Γ value (line 14). This ensures that all values, including the *timeout* have not been altered by an adversary; and (2) Verify that each of the q shares satisfy the puzzle (lines 15-17). If these verifications succeed, FraudSys waits until *timeout* expires to confirm the user action A, for posting by the online service (line 18). It then uses the time required by the device to solve the puzzle, to re-evaluate the hashrate of the device (lines 19-20). It updates the stored hashrate only if the new value is above a minimally accepted hashrate value (lines 20-21).

5.4.2 The Fraud Detection Module

To assign a fraud score to a user activity in real-time, the fraud detection module can only rely on the existing history of the user and of the subject on which the activity is performed. We propose an approach that builds on the *co-activity graphs* of subjects, where nodes correspond to user accounts that performed activities on the subject, and

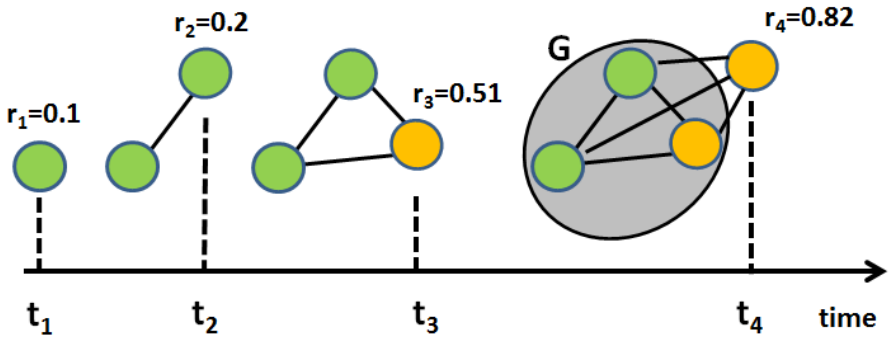


Figure 5.7: Fraud detection illustration: temporal evolution of the co-activity graph of a subject. The nodes represent user accounts that have performed an activity on the subject. Edges connect accounts with common past activities. As a new user account posts an activity, FraudSys assigns the activity a fraud score (the $r_1..r_4$ values), based on its connectivity to previous activities. Yellow nodes are considered fraudulent ($r > 0.5$).

edges connect nodes whose user accounts have a history of activities that targeted the same subjects. Edge weights denote the size of that history. Figure 5.6 shows the co-review (where activities are reviews) graph of a fraudulent Google Play app, that received fake reviews from 2 fraud workers. Each cluster is formed by accounts controlled by one of the workers.

The fraud detection module leverages the adversary model findings (§ 5.3.1) that a fraudster-controlled user account that performs a new activity on a subject, is likely to be well connected to the co-activity graph of the subject, or at least one of its densely connected sub-graphs. Figure 5.7 illustrates this approach: Let U be a user account that performs an activity A for a subject S at time T. Let $G = (V, E)$ be the co-activity graph of S before time T. Let $G_T = (V_T, E_T)$ be the new co-activity graph of S, that also includes U, i.e., $V_T = V \cup U$. Given U, S and G, FraudSys extracts the following features, that model the relationship of U with S:

• **Connectivity features**. The percentage of nodes in V to whom U is connected. The average weight of the edges between U and the nodes in V. The average weight of those edges divided by the average weight of the edges in E. This feature will

indicate if U increases or decreases the overall connectivity of G. The number of triangles in G_T that have U as a vertex. The average edge weight of those triangles.

• **Best fit connectivity features**. Since U may be controlled by one of multiple fraudsters who target S, U may be better connected to the subgraph of G controlled by that fraudster. Then, use a weighted min-cut algorithm to partition G into components $G_1, .., G_k$, such that any node in a component is more densely connected to the nodes in the same component than to the nodes in any of the other components. $G_1, .., G_k$ may contain user accounts controlled by different fraudsters, see Figure 5.6. Identify the component G_b, $b \in \{1, ..k\}$ to which U is the most tightly connected (according to the above connectivity features). Output the connectivity features between U and G_b.

• **Account based features**. The number of activities previously performed by U. The age of U: the time between U's creation and the time when activity A is performed on S. The *expertise* of U: the number of actions of U for subjects similar to S. Similarity depends on the online service, e.g., same category apps in Google Play, pages with similar topics in Facebook.

The Fraud Detection module trains a probabilistic supervised learning algorithm on these features and uses the trained model to output the probability that a given activity is fraudulent. We detail the performance of various algorithms, over data that we collected from Google Play and Facebook, in § 6.8.

Per-fraudster timeout. We exploit the ability of the fraud detection module to identify accounts controlled by the same fraudster, to further restrict fraud. Specifically, instead of storing a *timeout* timestamp for each user account, FraudSys can store a single *timeout* per detected fraudster. Thus, FraudSys will accumulate penalties in a single, per-fraudster account. This facilitates Claim 3.

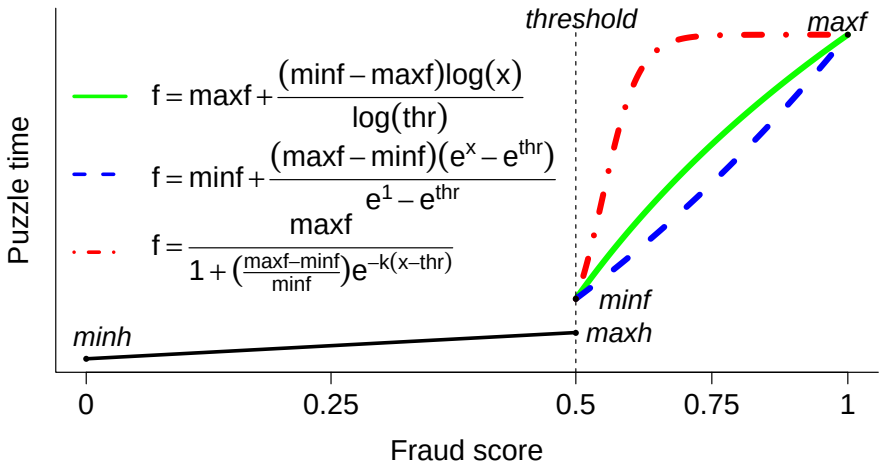


Figure 5.8: Comparison of functions to convert fraud scores (x axis) to time penalties (y axis). The logistic function (red dot-line) exhibits the required exponential increase.

5.4.3 The Fraud2Penalty Module

Given the fraud score r of an activity of user U (output by the Fraud Detection module), performed from a device D associated with the account of U (see the model section), the Puzzler module generates a puzzle whose difficulty is a function of both r and the computational capability of D. We now describe the Fraud2Penalty module, that converts r into a time penalty. We have explored several functions to convert the fraud score r of a user activity to a time penalty. Let $minh$ and $maxh$, and $minf$ and $maxf$, denote the minimum and maximum times imposed on the device from which an honest, respectively fraudulent activity is performed. Let thr denote the threshold fraud score above which we start to consider a user activity as being fraudulent. We propose a conversion function that increases linearly when $r < thr$, and exponentially when $r > thr$. Specifically, we propose a flexible generalization of

the logistic function (when $r > thr$), where the parameter k is the *growth rate*:

$$\begin{cases} \frac{maxh-minh}{thr} r + minh & 0 \leq r \leq thr \\ \frac{maxf}{1+(\frac{maxf-minf}{minf})e^{-k(r-thr)}} & thr \leq r \leq 1 \end{cases} \tag{5.4}$$

We have compared this logistic increase function with other functions, with the same linear increase in the honest region, but exponential ($(maxf - minf)\frac{e^r - e^{thr}}{e^1 - e^{thr}} + minf$) and logarithmic ($(minf - maxf)\frac{logr}{log(thr)} + maxf$) increase in the fraudulent regions. Figure 5.8 compares the logistic, exponential and logarithmic functions. It shows that unlike the exponential and logarithmic functions, the logistic function exhibits the desired rapid increase for fraud probability values above the threshold value. In § 6.8 we detail parameter values for the logistic conversion function,

5.4.4 The Hashrate Inference Module

New device registration. When a user registers a new device, the device sends its specs to the online service that forwards them to FraudSys. FraudSys leverages its list of profiled devices (see Table 5.2) to retrieve the hashrate of the profiled device with the most similar capabilities. FraudSys stores the new device along with this initial hashrate estimate under the id of the user that registers it (see the Puzzle module). Given this hashrate and the above time penalty, FraudSys uses Equation 5.3 to compute an initial puzzle difficulty.

Hashrate correction. The initial hashrate estimate of FraudSys may be incorrect. In addition, as discussed in the System Model, the user may be adversarial, thus attempt to provide an inaccurate view of the puzzle solving capabilities of his device. To address these problems, FraudSys employs an adaptive hashrate correction process. Specifically, an adversary with a more capable device than advertised (see e.g., Table 5.2) will solve the assigned puzzle faster. The incentive for this is a shorter wait time for his activity to post on the online service. If this occurs, FraudSys increases

its device hashrate estimate to reflect the observed shorter time required by the device to solve the puzzle (see Algorithm 2, lines 19-20).

5.5 FraudSys Properties

Claim 2 *A fraudster that performs a fraudulent activity with fraud score r from a device with hashrate η, is expected to compute $\frac{\eta \times maxf}{1+(\frac{maxf-minf}{minf})e^{-k(r-thr)}}$ double hashes.*

Proof. According to Equation 5.4, the time penalty assigned to a fraudulent activity with score r is $\tau = \frac{maxf}{1+(\frac{maxf-minf}{minf})e^{-k(r-thr)}}$. Then, Equation 5.3 ensures that the number of expected hashes that the device needs to perform to solve the puzzle of Equation 5.1 is $\eta \times \tau$, which concludes the proof. □

Let f be the number of fraud workers in the system (i.e., $f = |\mathcal{F}|$), τ be the average temporal penalty assigned by FraudSys to a fraudulent activity, and let p be the expected payout for a single fraudulent activity. We introduce then the following claim:

Claim 3 *FraudSys is a $(p/\tau, f/\tau)$-fraud preemption system.*

Proof. The best fit connectivity features of the Fraud Detection module (see § 5.4.2) enable FraudSys to detect activities performed from accounts controlled by the same fraudster. This, coupled with an extension of the *timeout* concept applied at the fraudster level (see § 5.4.2) ensures a serialization of fraudster activities. Then, the average number of fraudulent activities that a fraudster can post per time unit in FraudSys is $1/\tau$. This implies that, per time unit, the expected payout of a fraudster is p/τ, and a subject can be the target of at most f/τ fraudulent activities. This, according to the definition of § 5.3.2, completes the proof. □

5.5.1 Security Discussion and Limitations

The FraudSys puzzle not only ties the penalty computation to the user activity, but also addresses pre-computation, replay and guessing attacks: the adversary cannot predict the cookie value of its actions, thus cannot pre-compute puzzles and cannot reuse old cookies. It also offloads significant work from the FraudSys service, which no longer needs to keep track of puzzle assignments.

Device deception. An adversary with a specialized puzzle solving device (e.g., AntMiner) will be assigned puzzles with large difficulty values (see, e.g., Table 5.2), thus consume the same amount of time as when using a resource constrained device (e.g., a smartphone). The adversary can exploit this observation to avoid the implications of Claim 2: register a resource constrained device, but rely on a powerful back-end device to solve the assigned puzzles faster. The adversary has two options. First, report the solutions as soon as the back-end device retrieves them. In this case however, the adversary leaks his true capabilities, as FraudSys will update the adversary hashrate (Algorithm 2, line 20). Thus, subsequently, his assigned puzzles will have a significantly higher difficulty value. In a second strategy, the adversary estimates the time that his front-end device would take to complete the puzzle, then waits the remaining penalty time. In this case, the adversary incurs two penalties, the long wait time and the underutilized back-end device investment.

Adversary strategies: new user accounts. To avoid the implications of Claim 3, the adversary registers new user accounts. While new accounts are cheap, their freshness and lack of history will enable the account based features of the Fraud Detection module to label them as being likely fraudulent. As the adversary reuses such accounts, the connectivity features start to play a more important role in labeling their activities as fraudulent. Thus, the adversary has a small usable window of small penalties for new accounts.

While new honest accounts may also be assigned larger penalties for their first few activities, they will not affect the user experience: the user can continue her online activities, while her device solves the assigned puzzle in the background.

5.6 Empirical Evaluation

5.6.1 Datasets

We have collected the following datasets of fraudulent and honest behaviors from Google Play and Facebook.

Google Play: fraud behavior data. We have identified 23 workers in Freelancer, Fiverr and Upwork, with proven expertise on performing fraud on Google Play apps. We have contacted these workers and collected the ids of 2,664 Google Play accounts controlled by them. We have also collected 640 apps heavily reviewed from those accounts, with between 7 and 3,889 reviews, of which between 2% and 100% (median of 50%) were written from accounts controlled by the workers. These apps form our gold standard *fraud app* dataset. We have also collected the 23,028 fake reviews written from the 2,664 fraudster controlled accounts for the 640 apps. Figure 5.6 shows the co-review graph of one of these apps, that received fake reviews from 2 of the identified 23 workers.

Google Play: honest behavior data. We have selected 925 candidate apps that have been developed by Google designated "top developers". We have removed the apps whose apks (executables) were flagged as malware by VirusTotal. We have manually investigated 601 of the remaining apps, and selected a set of 200 apps that (i) have more than 10 reviews and (ii) were developed by reputable media outlets (e.g., NBC, PBS) or have an associated business model (e.g., fitness trackers). We call these the gold standard *benign app dataset*.

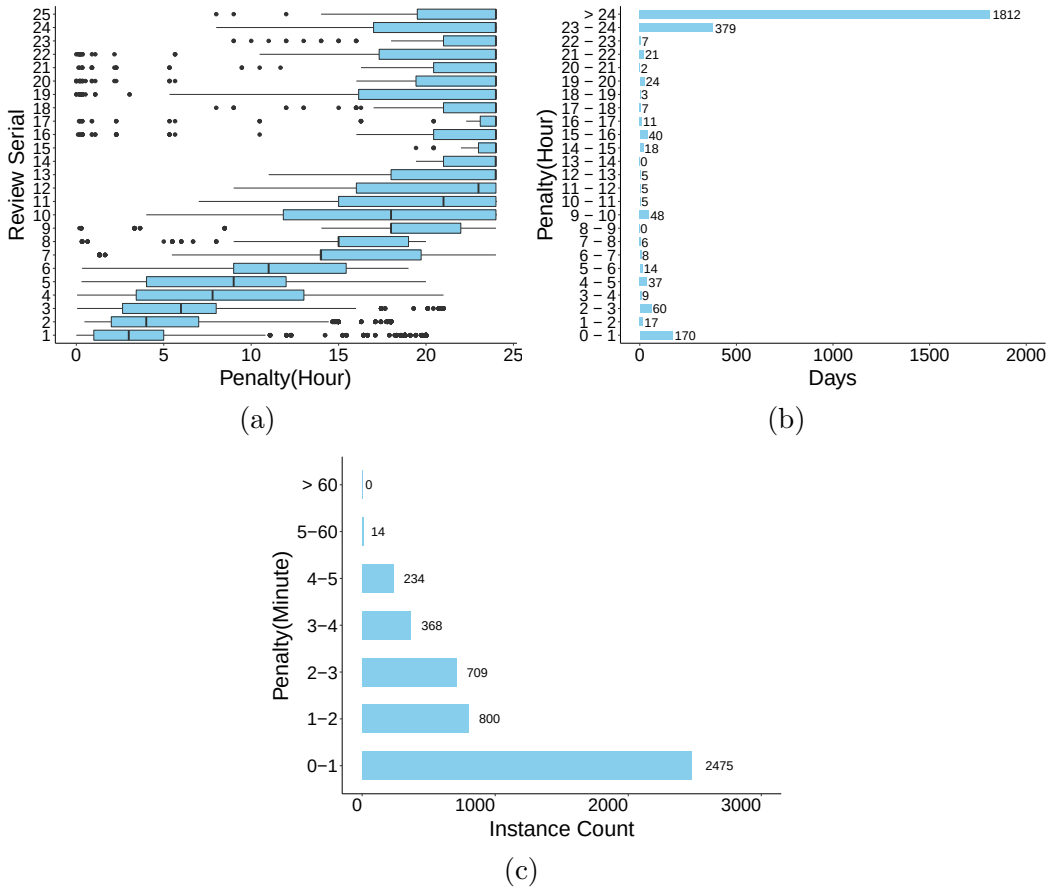


Figure 5.9: Stats over the Google Play data when $maxf = 24$h, $minh = 2$s, $maxh = minf = 5$min. (a) Evolution of average, 1st and 3rd quartile of the penalty imposed on the i-th fraud activity of a fraudster for the same subject. It shows a steep increase: the average penalty of the first three fraud activities for a subject sums to 15.34h, while the average penalty of the 12th activity exceeds 24h. (b) Distribution of per-fraudster daily penalties, over data from 23 fraudsters: in 1,812 days out of 2,708 days, the penalty assigned to a single fraudster exceeds 24 hours. (c) Distribution of penalties assigned to an honest review. Only 14 out of 4,600 honest review instances received a penalty exceeding 5 minutes, but still below 1 hour.

Device	Hashrate	Diff (5s)	(12hr)	(7 day)
Nexus 4	6.53 KH/s	16.32K	141.04M	1.97G
Nexus 5	13.26 KH/s	33.15K	286.41M	4.00G
LG Leon LTE	10.1 KH/s	25.25K	218.16M	3.05G
NVS 295	1.7MH/s	4.25M	36.72G	514.08G
Server	80 MH/s	200M	1.72T	24.19T
AntMiner	4.72 TH/s	11.8T	101.95P	1427P

Table 5.2: Hashrate profiling table for various device types (smartphone, tablet, PC and Bitcoin miner), along with difficulty values for penalty times of 5s, 12 hours and 7 days.

We have identified 600 reviewers of these 200 benign apps and 140 reviewers of the 640 fraud apps (see above), such that each has reviewed at least 10 paid apps, i.e., paid to install the app, then reviewed it, and had at least 5 posts on their associated Google Plus (social network) accounts. These 740 user accounts form our gold standard *honest user dataset*. We have then retrieved and manually vetted 854 reviews written by the 600 honest reviewers for the 200 benign apps, and 207 reviews written by the 140 honest reviewers of the 640 fraud apps. Each selected review is informative, containing both positive and negative sentiment statements. We call the resulting dataset, the *honest review dataset*, with 1,061 reviews.

Facebook Like dataset. We have used a subset of the dataset from [BSLL+16b], consisting of 15,694 Facebook pages, that each has received at least 30 likes. The pages were liked from 13,147 user accounts, of which 6,895 are fraudster controlled, and 6,252 are honest. In total, these fraudsters have posted 274,297 fake likes, and the honest accounts have posted 180,400 honest likes.

5.6.2 Device Hashrate Profile

We have profiled the hashrate of several devices, ranging from smartphones to a Bitcoin mining hardware (AntMiner S7: ARMv7 CPU, 254 Mb of RAM, 135 BM1385 chips @ 700MHz). Since Bitcoin mining requires capabilities far exceeding those of

Strategy	FPR%	FNR%	Accuracy%
k-NN	1.41	4.45	97.92
SVM	5.8	11.3	92.40
Random Forest	3.44	6.46	95.69

Table 5.3: 10-fold cross validation results of supervised learning algorithms in fraud vs. honest Google Play review classification. k-NN achieves the lowest FPR and FNR.

smartphones, we have implemented an Android app to evaluate the hashrate of several Android devices. Table 5.2 shows the hashrate values for the profiled devices, along with the corresponding difficulty (Δ) values for puzzles required to impose 5 second, 12 hour and 7 day time penalties on such devices. We observe the significant gap between the hashrate of a smartphone (10-15 KH/s) and a specialized device (4.72 TH/s). This motivates the need for the puzzles issued by FraudSys to have different Δ values for various user devices. FraudSys maintains a similar table in order to be able to build appropriate puzzles for newly registered devices.

5.6.3 Fraud Penalty Evaluation: Google Play

Supervised learning algorithm choice. We first used 10 fold cross-validation to evaluate the ability of the Fraud Detection module to correctly classify the 23,028 fraudulent vs. 1,061 honest reviews of the Google Play dataset previously described. Table 5.3 shows the false positive (FPR) and negative (FNR) rates, as well as the accuracy achieved by the top 3 performing supervised algorithms. k-NN has the lowest FPR and FNR, for an accuracy of 97.92%. Thus, in the following experiments we use only k-NN.

Parameter evaluation. We have used the fraud and honest review datasets described earlier, to compute the temporal penalties imposed by FraudSys on fraudsters and honest users. We have performed the following experiments. In each experiment,

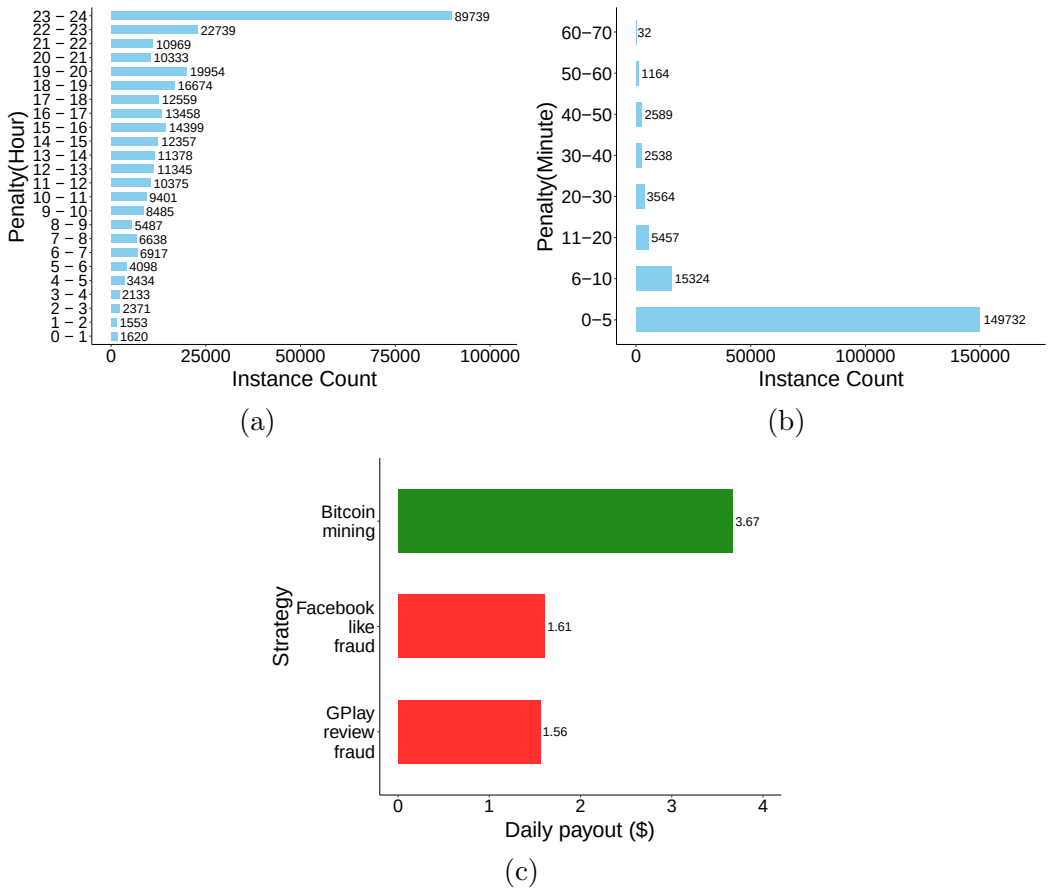


Figure 5.10: (a) Penalty distribution for the fake Facebook likes. 84% of the likes received a penalty that exceeds 12 hours, and the average fake like penalty is 19.32 hours. (b) Penalty distribution for the honest Facebook likes. 82.97% of the honest likes are assigned a penalty of under 5 min. The maximum penalty assigned to an honest like is 70 minutes. (c) Comparison of daily payouts provided by Bitcoin mining, writing fake reviews in Google Play and posting fake likes in Facebook, under FraudSys. Fraud does not pay off under FraudSys: the fraud payout is less than half the Bitcoin mining payout.

we use the data of 22 fraud workers and 200 randomly chosen honest reviews (out of 1,061) to train the supervised learning algorithm (k-NN) then test the model on the data of the remaining fraud worker and on the remaining 861 honest reviews. Thus, we have performed 23 experiments, one for each worker.

We set the $maxf$ parameter such that the average daily payout of a fraudster is below the average Bitcoin mining payout with a last generation AntMiner device. Thus, this ensures that even such a powerful adversary has more incentive to do Bitcoin mining instead of search rank fraud. Specifically, the above AntMiner's current (Jan. 2017) average daily payout is 0.0037 BTC. At the current BTC to USD rate, this means \$3.67 per day [1]. In addition, we have experimented with $maxf$ values ranging from 12 to 48 hours. The average penalty assigned by FraudSys to a fraudulent review is 8.01 hours when $maxf$=12h, 15.34h when $maxf$=24h, and 29.33h when $maxf$=48h. Figure 5.9(a) shows the median, first and third quartiles for the time penalty (in hours) imposed on the i-th fraudulent activity performed by a fraudster for a subject, when $maxf$= 24h: the 12th fake activity receives a median penalty of 24h.

Thus, we set $maxf$=24h, which is sufficient for Google Play reviews: A fraudster would be able to post on average less than 2 fake reviews per day, thus, even with a reward of \$2 per fraud activity (see Figure 5.3), achieve a payout of around \$3.15 per day, below the Bitcoin mining payout. In addition, we have set $minh = 2$s. Figure 5.1 shows the penalty timelines of two workers when $minh = 2$s, $maxh = minf = 5$ min, $maxf = 24$ hours, $thr = 0.5$, and $k = 30$ (for a steep increase of time penalty with fraud score). We note that a $maxh = 5$ min is not excessive: this penalty is not imposed on the user, but on his device. The user experience remains the same in the online service.

[1]Historically speaking, the BTC to USD rate is increasing. The next generation AntMiner coming up this year is expected to be 3 times more capable.

Each vertical bar shows the daily temporal penalty assigned to a single worker, over reviews posted from multiple accounts. The maximum daily penalty of the two workers is 1,247 hours and 3,079 hours respectively. We observe that each worker has many days with a daily penalty exceeding 24 hours.

Figure 5.9(b) shows for $maxh = minf = 5$min, the overall distribution of daily penalties assigned by FraudSys, over all the 23 fraud workers, in the above experiment. It shows that during most of the active days, fraud workers are assigned a daily penalty exceeding 24 hours. Figure 5.9(c) (also for $maxh = minf = 5$min) shows the distribution of per-review penalty assigned by FraudSys to honest reviews, shown over 4,600 (23×200) honest reviews. Irrespective of the $maxh$ value, only 14 honest reviews were classified as fraudulent, but assigned a penalty below 1 hour. We observed minimal changes in the distribution of penalties of fraudulent reviews when $maxh = minf$ ranges from 5 to 15 minutes.

5.6.4 Fraud Penalty Evaluation: Facebook

We have performed a similar parameter analysis using the Facebook "like" dataset. Since this dataset lacks information about the fraudsters who control the accounts that posted fake likes, we focus on the penalties assigned by FraudSys to fake and honest likes.

Figure 5.10(a) shows the distribution of penalties assigned to fake likes and Figure 5.10(b) shows the distribution of the honest likes. Compared to the results over the Google Play data, we observe a higher FPR, i.e., more honest likes with fraud level penalties. We posit that this is due to the fewer features that we can extract for the Facebook likes, as, unlike for Google Play reviews, we lack the time of the activity. Specifically, absence of like sequence information enables us to only extract

features based on the last "snapshot" of the page, and not the current page snapshot when the like was posted.

However, 82.97% of the honest likes receive a penalty of under 5 mins and the maximum penalty assigned to an honest review is 70 mins. In addition, 84% of the fake likes receive a penalty that exceeds 12 hours, and the average penalty for a fake like is 19.32 hours. Figure 5.10(c) compares the daily payouts received by an AntMiner equipped fraudster who writes fake reviews in Google Play (at $1 per fake review), posts fake likes (at $2 per fake like), or honestly uses his device to mine Bitcoins. It shows that under FraudSys, fraud doesn't pay off: the Bitcoin mining payout is more than double the fraud payout for either fake reviews or likes.

5.7 Research Contributions Acknowledgment

It is a pleasure to acknowledge the contributions of my co-authors; Ruben Recabarren, Dr. Dongwon Lee and my supervisor Dr. Bogdan Carbunar for their helpful support during this research. Ruben has made substantial contributions to conception and design puzzle cookie solution. My special gratitude goes to Dr. Carbunar and Dr. Lee for his contribution to the design the research and critical feedback to shape the manuscript.

5.8 Summary

We have introduced the concept of real-time fraud preemption systems, named as the FraudSys, that seek to restrict the profitability and impact of fraud in online systems. We propose and develop stateless, verifiable computational puzzles, that impose minimal overheads, but enable their efficient verification. We have developed a graph based, real-time algorithm to assign fraud scores to user activities and mechanisms to

convert scores to puzzle difficulty values. We used data collected from Google Play and Facebook to show that our solutions impose significant penalties on fraudsters, and make fraud less productive than Bitcoin mining.

CHAPTER 6

Search Rank Fraud De-Anonymization in Online Systems

6.1 Motivation

The developers of top ranking products in peer-review sites like Google Play, Amazon, or Yelp receive higher rewards, that include direct payments and ad-based revenues. Statistics maintained by peer-review sites concerning user activities for a product (e.g., reviews, ratings, likes, followers, app install counts) are known to play an essential part in the product's ranking [Pos12, Luc11, Ank13]. This has created a black market for *search rank fraud*, mediated by an abundance of crowdsourcing sites, e.g., [Fiv, Upw, Fre, Zee, Peo]. Specifically, crowdsourcing fraudsters create or purchase hundreds of user accounts in the peer-review site, then post activities for the products of developers who hire them, from the accounts they control.

Discouraging search rank fraud is essential to ensure trust in peer-review sites and the products that they host. Previous work in this area has focused on fraud detection [CHZY17, RRCC16b, MLG12, WGF17, FML+13b, MKL+13b, LFW+17b, HSB+16a, BSLL+16c, ACF13a, HSB+16c, RA15a, ACF13b]. Most peer-review sites filter out detected fraudulent activities [Cip16, Per16, MVLG13]. However, a study with 58 fraudsters that we recruited from 6 crowdsourcing sites revealed that workers with years of search rank fraud expertise are actively contributing to such jobs, and are able to post hundreds of reviews for a single product at prices ranging from a few cents to $10 per review. This suggests that fraud detection alone is unable to prevent large scale search rank fraud.

In this paper we propose a new approach to discourage search rank fraud. We introduce the *fraud de-anonymization* problem, that aims to attribute detected search rank fraud in a peer-review site, to the crowdsourcing site fraudsters who posted it. Further, to understand and model search rank fraud behaviors, we have developed

a questionnaire and used it to survey 58 fraudsters recruited from 6 crowdsourcing sites. We have collected data from search rank fraud jobs and worker accounts in Upwork. We have collected a gold standard dataset of 956 user accounts in Google Play, attributed to 23 crowdsourced workers. We have developed a guilt-by-association process to expand this dataset with another 1,308 user accounts, for a total of 2,664 fraudster-attributed accounts. We analyze the activities performed from these accounts in the wild. We observe and report several adversary traits, including the existence of an *expert core* of fraudsters, who can control hundreds of user accounts and post tens of daily reviews for a single product, can change their behaviors to avoid detection (e.g., to throttle their daily review activities and dilute them over long time intervals), can be rehired to promote the same product at later times, and that products can be fraudulently promoted by multiple fraudsters.

We leverage the identified traits to introduce DOLOS[1] a system that cracks down fraud by unmasking the human masterminds responsible for posting significant fraud. DOLOS integrates search rank fraud detection with *fraud attribution* to reveal the lurking organized activities that power the fraud, and pinpoint their human command centers. DOLOS detects then attributes fraudulent user accounts in the online service, to the crowdsourcing site accounts of the workers who control them. We devise MCDense, a min-cut dense component detection algorithm that analyzes common activity relationships between user accounts to uncover groups of accounts, each group controlled by a different search rank fraudster. We further leverage stylometry and supervised learning to attribute MCDense detected groups to the crowdsourcing fraudsters who control them.

DOLOS correctly attributed 95% of the reviews of 640 apps (that received significant, ground truth search rank fraud) to their authors. For 97.5% of the apps, DOLOS correctly de-anonymized at least one of the fraudsters who authored their fake

[1]DOLOS is a concrete block used to protect harbor walls from erosive ocean waves.

reviews. DOLOS achieved 90% precision and 89% recall when attributing the above 2,664 fraudulent accounts to the fraudsters who control them. Further, MCDense significantly outperformed an adapted densest subgraph solution.

We have evaluated DOLOS on 13,087 Google Play apps (and their 820,760 reviews) that we monitored over more than 6 months. DOLOS discovered that 1,056 of these apps have suspicious reviewer groups. Upon close inspection we found that (1) 29.9% of their reviews were *duplicates* and (2) 73% of the apps that had at least one MCDense discovered clique, received reviews from the expert core fraudsters that we mentioned above. We also report cases of *fraud re-posters*, accounts who re-post their reviews, hours to days after Google Play filters them out (up to 37 times in one case).

To evaluate MCDense, we introduce two coverage scores, p-coverage and p-SCC, that measure the quality of detected community partitions. We adapt dense subgraph detection [Tso15] and loopy believe propagation [ACF13a] solutions to the fraud de-anonymization problem. We show that MCDense consistently and significantly outperforms DSG on both coverage scores. While LBP can be used to accurately detect fraud, it cannot determine if all the accounts detected as fraudulent are controlled by a single or multiple workers.

In summary, we introduce the following contributions:

- **Fraud de-anonymization problem formulation**. Introduce a new approach to combat and discourage search rank fraud in peer-review sites.
- **Study and model search rank fraud**. Survey 58 fraudsters from 6 crowdsourcing websites on fraud posting capabilities and behaviors. Collect search rank fraud jobs posted on Upwork and analyze common bidding and winning behaviors between fraud workers. Collect gold standard fraudster-attributed Google Play user accounts and study their behaviors in the real world. Extract and present fraudster behaviors traits.
- **Dolos**. Exploit extracted insights to develop the first fraud de-anonymization

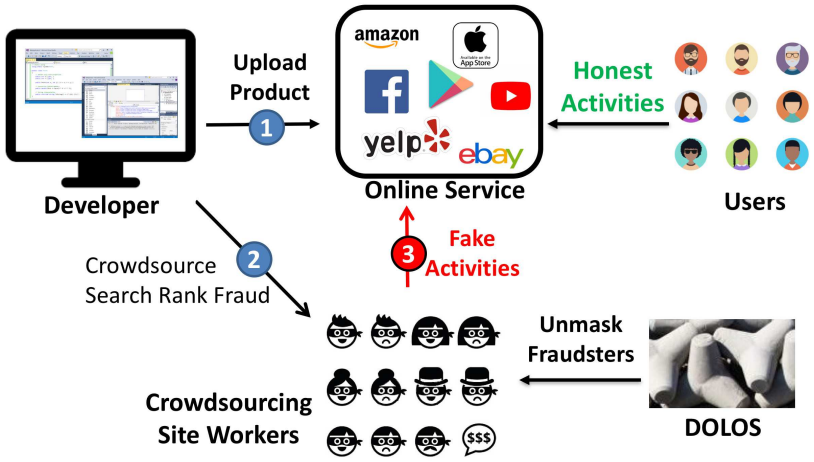


Figure 6.1: System and adversary model. Developers upload products, on which users post activities, e.g., reviews, likes. Adversarial developers crowdsource search rank fraud. Unlike fraud detection solutions, Dolos unmasks the human fraudsters responsible for posting search rank fraud.

system. Devise MCDense, a min-cut dense component detection algorithm to identify accounts controlled by the same fraudster. Use stylometry to attribute detected components to the profiles of known crowdsourcing workers.

- **Evaluation**. Evaluate Dolos extensively on Google Play data. Identify orthogonal evidence of fraud from detected suspicious products. Develop novel community coverage scores. Show that MCDense significantly outperforms adapted dense subgraph and loopy believe propagation solutions, on the developed scores.
- **Open source**. The Dolos and MCDense code is available for download online [Dol].

6.2 System and Adversary Model

We consider an ecosystem that consists of peer-review sites and crowdsourcing sites. Peer-review sites host accounts for developers, products and users, see Figure 6.1. Developers use their accounts to upload products. Users post *activities* for products,

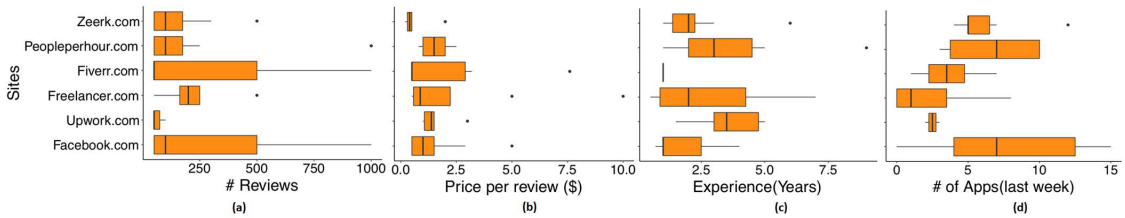


Figure 6.2: Statistics over 44 fraudsters (targeting Google Play apps) recruited from 5 crowdsourcing sites: minimum, average and maximum for (a) number of reviews that a fraudster can write for an app, (b) price demanded per review, (c) years of experience, (d) number of apps reviewed in the past 7 days. Fraudsters report to be able to write hundreds of reviews for a single app, have years of experience and are currently active. Prices range from 56 cents to $10 per review.

e.g., reviews, ratings, likes, installs. Product accounts display these activities and statistics, while user accounts list the products on which users posted activities.

Users register mobile devices to their accounts, then install apps on them. Users can only review apps that they have previously installed. Reviews have a star rating (1-5) and a text component.

The survival of mobile apps in Google Play is contingent on their search rank. Higher ranked apps are installed more frequently and generate more revenue, either through ads or direct payments. While Google keeps their ranking algorithm secret, popular belief (e.g., [Ank13]) holds that large numbers of positive reviews help new apps achieve higher search rank.

Crowdsourcing sites host accounts for *workers* and *employers*. Worker accounts have unique identifiers and bank account numbers used to deposit the money that they earn. Employers post *jobs*, while workers *bid* on jobs, and, following negotiation steps, are assigned or *win* the jobs.

We consider product developers who hire workers from crowdsourcing sites, to perform search rank fraud. In this paper we focus on workers who control multiple user accounts in the online system, which they use to post fake activities, e.g., review, rate, install. We study such workers in Section 6.7.

6.3 The Fraud De-Anonymization Problem

Let $\mathcal{W} = \{W_1, .., W_n\}$ be the set of crowdsourcing worker accounts. Let $\mathcal{U} = \{U_1, .., U_m\}$ be the set of user accounts and let $\mathcal{A} = \{A_1, .., A_a\}$ be the set of products hosted by the online service, respectively. We define the fraud de-anonymization problem as follows:

Fraud De-Anonymization Problem. Given a product $A \in \mathcal{A}$, return the subset of fraudsters in $\mathcal{W}$ who control user accounts in $\mathcal{U}$ that posted activities for A.

Unlike standard de-anonymization, which refers to the adversarial process of identifying users from data where their Personally Identifiable Information (PII) has been removed, the fraud de-anonymization problem seeks to attribute detected search rank fraud to the humans who posted it.

A solution to this problem will enable peer-review sites to (1) put a face to the humans who post fraud for the products that they host, i.e., identify their banking information and use it to pursue fraudsters, and (2) provide proof of fraud to customers, e.g., through links to the crowdsourcing accounts responsible for fraud posted on products they browse. Thus, fraud de-anonymization may provide counter-incentives both for the crowdsourcing workers who participate in fraud jobs, and for the product developers who recruit fraudsters.

6.4 A Study of Search Rank Fraud

In this section we describe our efforts to understand and model search rank fraud workers. Succinctly, we have (1) performed a user study with fraud workers recruited from several crowdsourcing sites, (2) collected and analyzed search rank fraud data from Upwork, (3) collected a gold standard set of user accounts, attributed to a fraudster identified from a crowdsourcing site and (4) analyzed the behaviors exhibited by these user accounts. We have developed our protocols to interact with participants

and collect data in an IRB-approved manner (Approval #: IRB-15-0219@FIU and IRB-18-0077@FIU).

In the following we describe each contribution. We use the terms *worker*, *fraudster* and *fraud worker*, interchangeably.

6.4.1 Motivation: Fraudster Capabilities

To evaluate the magnitude of the problem, we have first contacted 44 workers from several crowdsourcing sites including Zeerk (12), Peopleperhour (9), Freelancer (8), Upwork (6) and Facebook groups (9), who advertised search rank fraud capabilities for app markets. We asked them (1) how many reviews they can write for one app, (2) how much they charge for one review, (3) how many apps they reviewed in the past 7 days, and (4) for how long they been active in promoting apps.

Figure 6.2 shows statistics over the answers, organized by crowdsourcing site. It suggests significant profits for fraudsters, who claim to be able to write hundreds of reviews per app (e.g., an average of 250 reviews by Freelancer workers) and charge from a few cents ($0.56 on average from Zeerk.com workers) to $10 per review (Freelancer.com). Fraudsters have varied degrees of expertise in terms of years of experience and recent participation in fraud jobs. For instance, in recently emerged Facebook groups, that either directly sell reviews or exchange reviews, fraudsters have less than 2.5 years experience, but are very active, with more than 7 jobs in the past 7 days on average, and are economical ($1.3 on average per review). Further, fraudsters from Peopleperhour and Upwork have more than 2.5 years experience and more than 3 recent jobs on average.

Subsequently, we have developed a more detailed questionnaire to better understand search rank fraud behaviors and delivered it to 14 fraud freelancers that we recruited from Fiverr. We paid each participant $10, for a job that takes approx. 10

minutes. The IPs from which the questionnaire was accessed revealed that the participants were from Bangladesh (5 participants), USA (2), Egypt (2), Netherlands, UK, Pakistan, India and Germany (1). The participants declared to be male, 18 - 28 years old, with diverse education levels: less than high school (1 participant), high school (2), associate degree (3), in college (5), bachelor degree or more (3).

The participants admitted an array of fraud expertise (fake reviews and ratings in Google Play, iTunes, Amazon, Facebook and Twitter, fake installs in Google Play and iTunes, fake likes and followers in Facebook and Instagram, influential tweets in Twitter). With a focus on search rank fraud targeting Google Play apps, we found a mix of (1) inexperienced and experienced fraudsters: 4 out of 14 had been active less than 2 months and 6 fraudsters had been active for more than 1 year, and (2) active and inactive fraudsters: 4 had not worked in the past month, 9 had worked on 1-5 fraud jobs in the past month, and 1 worked on more than 10 jobs; 8 fraudsters were currently active on 1-5 fraud jobs, and 1 on more than 5.

Further, we observed varying search rank fraud capabilities when it comes to the magnitude of the fraud on a per-app level. For instance, 1 fraudster wrote at most 1 review per app, 2 wrote 2-5 reviews, 7 fraudsters said that they wrote between 5 to 50 reviews per app, while 1 wrote 51 to 100 reviews. 1 fraudster performed less than 10 installs per job, 3 had 11 to 100 installs, 3 had 101 to 1,000 installs per job, while 1 fraudster said that he performed more than 1,000 installs for a single app. 8 fraudsters said that they have access to more than 10 mobile devices, with 1 having more than 50 devices.

Of the 14 fraudsters surveyed, 3 admitted to working in teams that had more than 10 members, and to sharing the user accounts that they control, with others. 10 fraudsters said that they control more than 5 Google Play accounts and 1 fraudster had more than 100 accounts. Later in this section we show that this is realistic, as other 23 fraudsters we recruited, were able to reveal between 22 and 86 Google Play

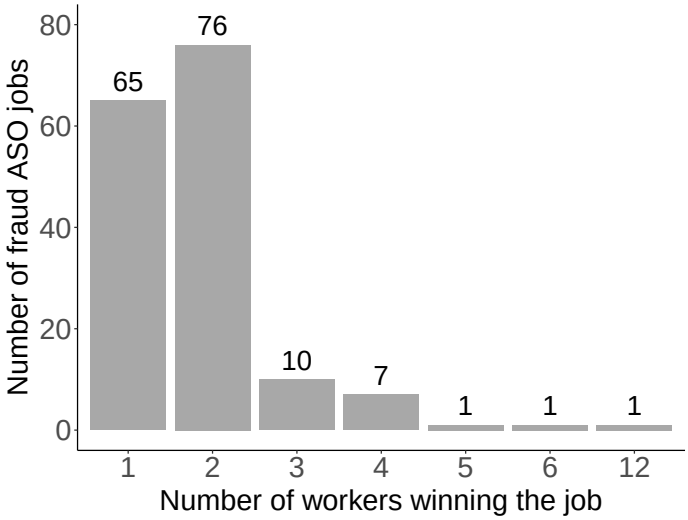


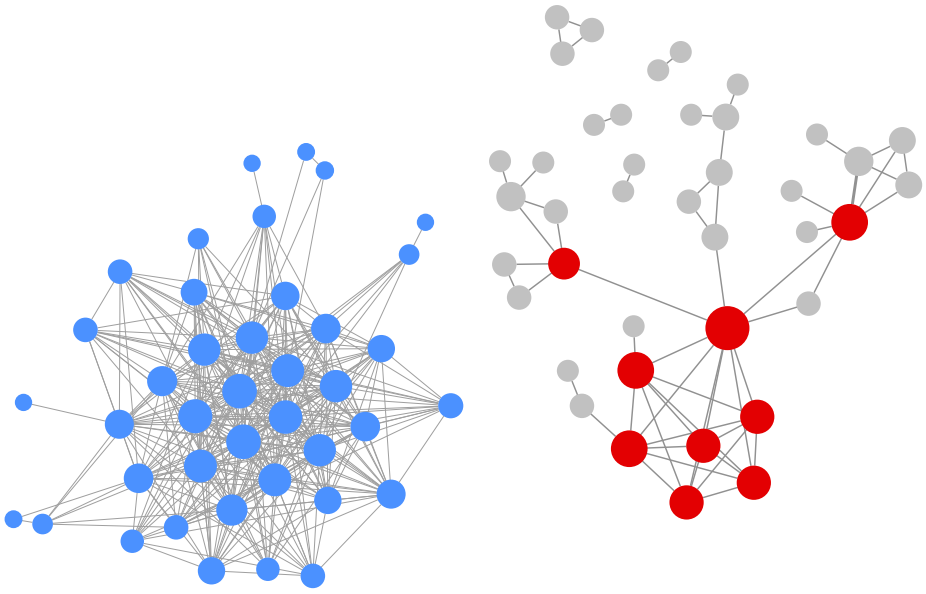
Figure 6.3: Distribution of winning workers for search rank fraud jobs: developers hire multiple workers. More jobs are assigned to 2 or more workers than to 1. This reveals the need for DOLOS to detect fraudulent communities and attribute them to different fraud workers.

accounts that they control. Further, 4 fraudsters said that they never abandon an account, 5 said that they use each account until they are unable to login, and 4 said that they use it for at most 1 year. This is confirmed by our empirical observation of the persistence of fraud (see end of section 6.4.3).

6.4.2 A Study of Search Rank Fraud Jobs

We identified and collected data from 161 search rank fraud jobs in Upwork that request workers to post reviews on, or install Google Play and iTunes apps. We have collected the 533 workers who have bid on these jobs. We call the bidding workers that are awarded a job, *winners*. To achieve this, we have developed a Python crawler to collect data both from crowdsourcing sites and from Google Play.

Figure 6.3 shows the distribution of the number of winners per search rank fraud job. One job of the 161, was awarded to 12 workers; more jobs were awarded to 2 workers than to only 1. This indicates that hiring multiple workers is considered bene-



(a) (b)

Figure 6.4: (a) Worker co-bid graph: Nodes are Upwork workers. An edge connects two workers who co-bid on search rank fraud jobs. We see a tight co-bid community of workers; some co-bid on 37 jobs. (b) Worker Co-win graph with an expert core of 8 workers (red), each winning $8-15$ jobs. Edges connect workers who won at least one job together. Any two workers collaborated infrequently, up to 4 jobs. Dolos exploits these observations to detect fraudulent account communities controlled by different fraudsters.

ficial by adversarial developers, and suggests the need to attribute detected organized fraud activities to human masterminds (see next section).

In order to understand the extent to which crowdsourced workers participate in common on search rank fraud jobs, we introduce the concepts of *co-bid* and *co-win graphs*. In the co-bid graph, nodes are workers who bid on fraud jobs; edges connect workers who bid together on at least one job. The edge weights denote the number of jobs on which the endpoint workers have bid together. In the co-win graph, the weight of an edge is the number of fraud jobs won by both endpoint workers.

Out of the 56 workers who won the 161 jobs, only 40 had won a job along with another bidder. Figure 6.4(a) shows the co-bid graph of these 40 winners, who form a

Figure 6.5: Attributed, fraudster-controlled accounts. The numbers of Google Play accounts revealed by the detected fraudsters are shown in red. Each of the 23 fraudsters has revealed between 22 to 86 accounts. Guilt-by-association accounts are shown in orange. We have collected a total of 2,664 accounts (red + orange). One fraudster controls (at least) 217 accounts.

tight community. Figure 6.4(b) plots the co-win graph of the 40 winners. We observe an "expert core" of 8 workers who each won between 8 to 15 jobs. Further, we observe infrequent collaborations between any pair of workers: any two workers collaborated on at most 4 jobs.

6.4.3 Fraudster Profile Collection (FPC)

We have collected a first gold standard dataset of attributed, fraudster controlled accounts in Google Play. For this, we have identified and contacted 100 Upwork, Fiverr and Freelancer workers with significant bidding activity on search rank fraud jobs targeting Google Play apps. Figure 6.5 shows the number of accounts (bottom, red segments) revealed by each of 23 most responsive of these workers: between 22 and 86 Google Play accounts revealed per worker, for a total of 956 user accounts.

Fraud app dataset. To expand this data, we collected first a subset of 640 apps that received the highest ratio of reviews from accounts controlled by the above 23 expert core workers to the total number of reviews. We have monitored the apps over a 6 months interval, collecting their new reviews once every 2 days. The 640 apps

Algorithm	Precision	Recall	F-measure
RF	95.5%	91.6%	93.5%
SVM	**98.5%**	**98.3%**	**98.5%**
k-NN	97.1%	96.4%	96.7%
MLP	98.6%	98.1%	98.4%

Table 6.1: Account attribution performance on gold standard fraudster-controlled dataset, with several supervised learning algorithms (parameters $d = 300$, $t = 100$, $\gamma = 80$, and $w = 5$ set through a grid search). SVM performed best.

had between 7 to 3,889 reviews. Half of these apps had at least 51% of their reviews written from accounts controlled by the 23 fraudsters. In the following we refer to these, as the *fraud apps*.

Union fraud graph. We have collected the account data of the 38,123 unique reviewers (956 of which are the seed accounts revealed by the 23 fraudsters) of the fraud apps, enabling us to build their *union fraud graph*: a node corresponds to an account that reviewed one of these apps (including fraudster controlled and honest ones), and the weight of an edge denotes the number of apps reviewed in common by the accounts that correspond to the end nodes. We have removed duplicates: an account that reviewed multiple fraud apps has only one node in the graph. The union fraud graph has 19,375,550 edges and 162 disconnected components, of which the largest has 37,566 nodes.

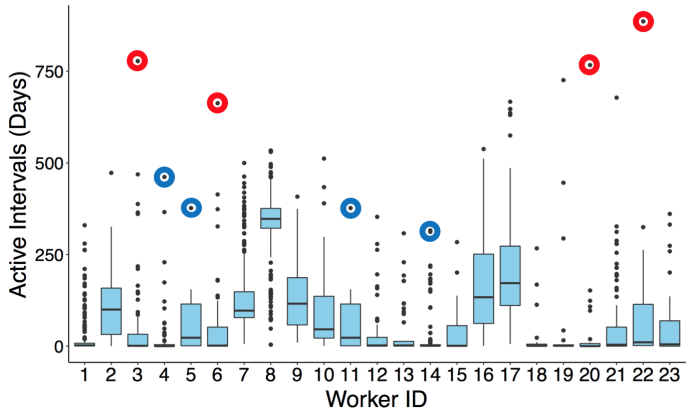
Guilt-by-association. We have labeled each node of the union fraud graph with the ID of the fraudster controlling it or with "unknown" if no such information exists. For each unknown labeled node U, we decide if U is controlled by one of the fraudsters, based on how well U is associated with accounts controlled by the fraudster. However, U may be connected to the accounts of multiple fraudsters (Trait 3, see Section 6.4.5).

To address this problem, we leveraged Trait 4 (see Section 6.4.5) to observe that random walks that start from nodes controlled by the same fraudsters are likely to share significant context, likely different from the context of nodes controlled by other fraudsters, or that are honest. We have pre-processed the union fraud graph

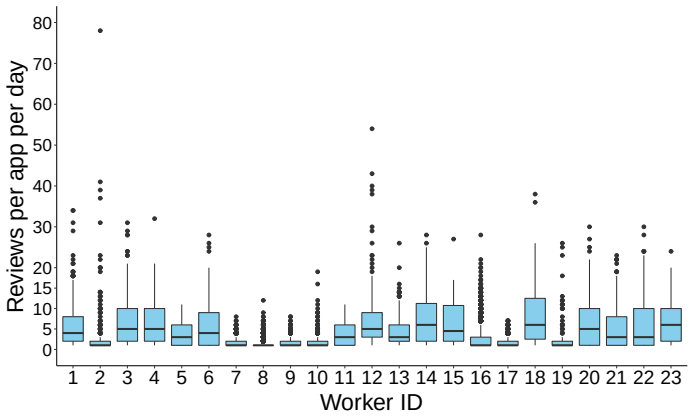
to convert it into a non-weighted graph: replace an edge between nodes u_i and u_j with weight w_{ij}, by w_{ij} non-weighted edges between u_i and u_j. We then used the DeepWalk algorithm [PARS14] to perform γ random walks starting from each node v in this graph, where a walk samples uniformly from the neighbors of the last vertex visited until it reaches the maximum walk length (t). The pre-processing of the union graph ensures that the probability of DeepWalk at node u_i to choose node u_j as next hop, is proportional to w_{ij}. DeepWalk also takes as input a window size w, the number of neighbors used as the context in each iteration of its SkipGram component. Deepwalk returns a d-dimensional representation in $\mathbb{R}^d$ for each of the nodes. We then used this representation as predictor features for the "ownership" of the account U - the fraudster who controls it.

Table 6.1 highlights precision, recall, and F-measure achieved by different supervised learning algorithms. We observe that SVM reaches 98.5% F-measure which suggests DeepWalk's ability to provide useful features and assist in our guilt-by-association process. We then applied the trained model to the remaining and unlabeled accounts in the union fraud graph obtaining new guilt-by-association accounts for each of the 23 workers. Figure 6.5 shows the number of seed and guilt-by-association accounts uncovered for each of the 23 fraudsters. We have collected $1,708$ additional accounts across workers for a total of 2,664 accounts.

Persistence of fraud. After more than 1 year following the collection of the 2,664 fraudster-controlled accounts, we have re-accessed the accounts. We found that 67 accounts had been deleted and 529 accounts were inactive, i.e., all information about apps installed, reviewed, +1'd was removed. 2,068 accounts were active. This is consistent with the findings from our fraudster survey, where 4 out of 14 surveyed fraudsters said that they never abandon an account, 5 said that they use each account until they are unable to login, and 4 said that they use it for at most 1 year. This suggests the limited ability of Google Play to block fraudster-controlled accounts.



(a)



(b)

Figure 6.6: (a) Per-fraudster distribution of active intervals. Each point represents the length of the active interval of the corresponding fraudster for a Google Play app that he has targeted. The boxes show the first and third quartiles of each fraudster, along with the median. The red dots correspond to a Google Play app, while the blue dots correspond to another app. Each of these apps was targeted by 4 of the 23 workers. These workers have the longest active interval for these apps. (b) Distributions of number of daily reviews posted by each worker per app. Several fraudsters stand out: fraudsters 2, 12, 43 and 18 provide an average of 38 to 78 daily reviews per each app they have targeted. Both plots were built on 2,835 apps which received more than 10 reviews from the 23 fraudsters.

6.4.4 Analysis of Fraud Behaviors

We study the activities performed from the accounts controlled by the above 23 fraud workers, in Google Play. For this, we have selected the 2,835 apps that have received at least 10 reviews from the 2,664 accounts controlled by the 23 fraudsters. We perform our analysis on these apps.

Active intervals. First, we study the *active interval* length of a worker for an app: the time interval between the first and last reviews posted from accounts controlled by the worker, for the app. Figure 6.6(a) shows the per-fraudster distribution of active interval durations, for the above 2,835 apps. We observe several apps (e.g., shown with red and blue circles) that were targeted by each of several fraudsters, over long time intervals (e.g., 1-2 years). We posit that workers may be rehired several times over the years, to perform search rank fraud. We revisit this hypothesis shortly.

Daily review capabilities. Second, we study the number of reviews that a worker has been able to post in a single day for a single app, from all the user accounts it controls. Figure 6.6(b) shows the distribution of the number of daily reviews posted by each of 23 fraudsters, per each app they target. It shows that several workers had days when they were able to post more than 38 reviews per day for one app. The second worker posted 78 reviews in a day for one app! These results corroborate the findings of our fraud worker survey described in Section 6.4.1.

Active intervals vs. reviews per active day. Figure 6.7 plots 3,369 data points, each representing an (app, fraudster) pair: the x axis value shows the fraudster's active interval for the app, and the y axis shows the average number of reviews that the fraudster has posted for that app, per active day. The number of points (3,369) is larger than the number of apps (2,835) since some apps were targeted by multiple fraudsters thus contribute multiple points. To improve visibility, we grouped the points into hexagonal-shaped bins, where the color of a bin is a function of the

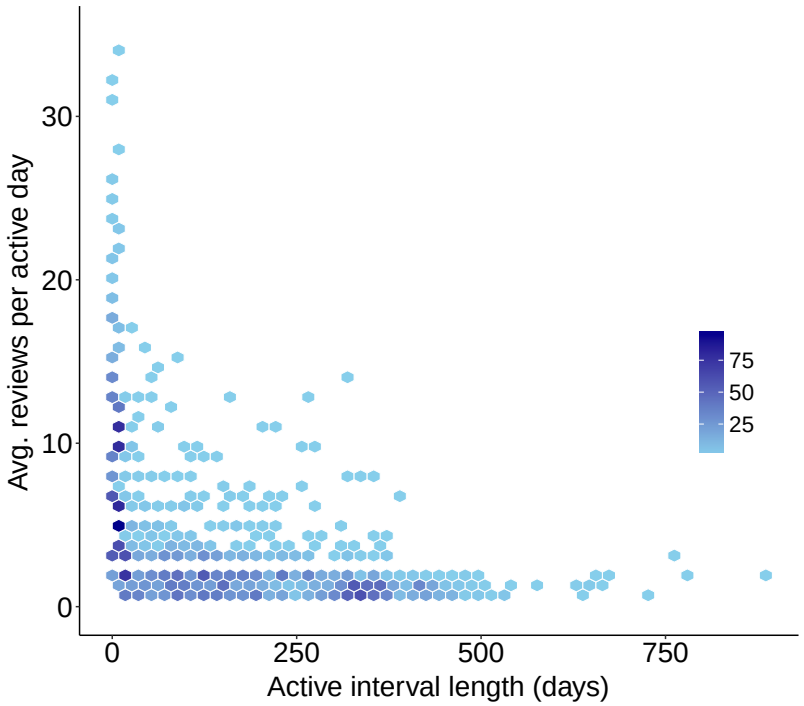


Figure 6.7: Scatterplot of the active interval length (x axis) vs. the average number of reviews per active day (y axis). Each point represents an (app, fraudster) pair: the x axis value shows fraudster's active interval for the app and the y axis shows the average number of reviews that the fraudster has posted for that app, per active day. Search rank fraud workers often prefer to "dilute" their reviews over a large number of days, instead of posting large number of reviews over a few days.

number of points it contains.

The scatterplot shows that the fraudsters who post a high number of reviews on average per active day (e.g., 18-34), tend to target apps only for a short time span (small active interval length), i.e., over 1-2 days. However, these points account for only 1.6% of the data. 75% of the data points plotted correspond to active intervals of up to 250 days. 64% of these points correspond to (app, fraudster) pairs where the fraudster wrote an average of 1 - 3 reviews per active day, 18% to 4 - 7 reviews per day and 18% to 18-34 reviews per day. 25% (858) of the data points correspond to active intervals of between 250 and 887 days. 72.12% of the points correspond to (app, fraudster) pairs where the fraudster wrote an average of 1 - 3 reviews per active day.

We observe that search rank fraud workers often "dilute" their reviews over a large number of days, instead of posting large number of reviews over only a few days. We believe that this is due to job requirements, which have evolved to avoid obvious defenses employed by peer-review systems, e.g., through detection of review spikes.

6.4.5 Empirical Adversary Traits

We summarize now several search rank fraudster traits suggested by our studies:

- **Trait 1**: Fraudsters control multiple user accounts which they use to perpetrate search rank fraud.
- **Trait 2**: While fraudsters have diverse search rank fraud capabilities, crowdsourcing sites have an "expert core" of successful search rank fraud workers. Many fraudsters are willing to contribute, but few have the expertise or reputation to win such jobs.

- **Trait 3**: Search rank fraud jobs often recruit multiple workers. Thus, targeted products may receive fake reviews from multiple fraudsters. This suggests that in addition to identifying fraudulent reviews, we need to further attribute them to their authors.
- **Trait 4**: Any two fraudsters collaborate infrequently, when compared to the number of search rank fraud jobs on which they have participated, see Figure 6.4(b).
- **Trait 5**: Fraudsters and the people who hire them evolve to avoid detection. For instance, search rank fraud workers are able to "dilute" their reviews over a large number of days, instead of posting large number of reviews over only a few days.
- **Trait 6**: Fraudsters may be rehired by the same product developer to promote the same product, several times over the years.
- **Trait 7**: Fraudsters, including experts, are willing to share information about their behaviors, perhaps to convince prospective employers of their expertise.

DOLOS exploits these traits to detect and attribute groups of fraudulent user accounts to the fraudsters who control them. While we do not claim that the sample data from which the traits are extracted is representative, in the evaluation section we show that DOLOS can accurately de-anonymize fraudsters.

6.5 Fraud De-Anonymization System

6.5.1 Solution Overview

We introduce DOLOS, the first fraud de-anonymization system that integrates activities on both crowdsourcing sites and online services. As illustrated in Figure 6.8, DOLOS (1) proactively identifies new fraudsters and builds their profiles in crowd-

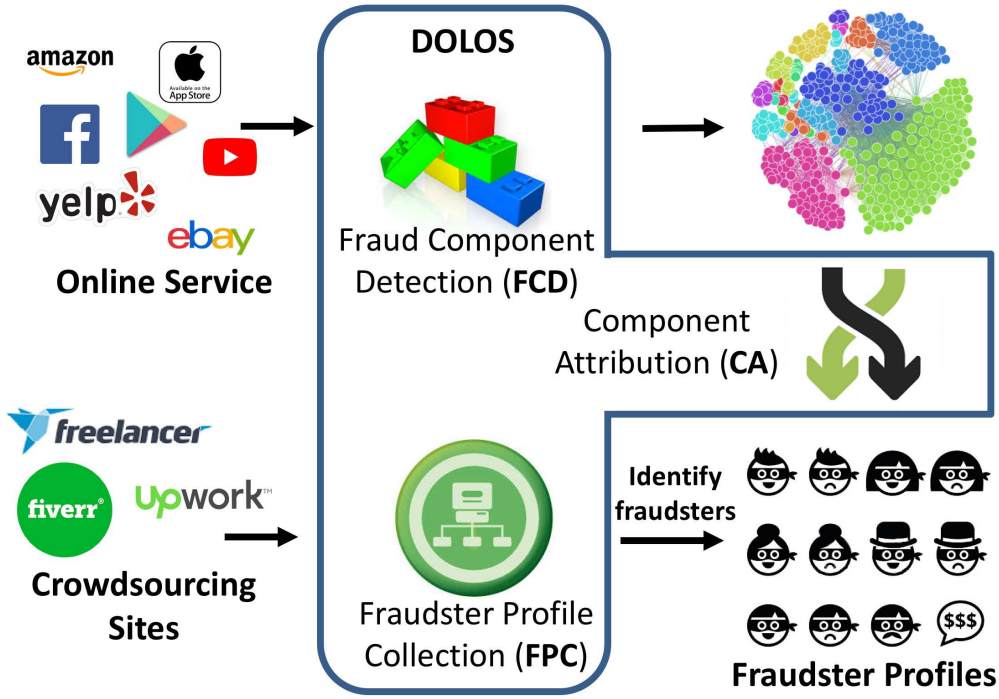


Figure 6.8: Dolos system architecture: The Fraud Component Detection (FCD) module partitions the co-activity graphs of apps into loosely inter-connected, dense components. The Component Attribution (CA) module attributes FCD detected components to fraudster profiles collected by the Fraudster Profile Collector (FPC).

sourcing sites, then (2) processes product and user accounts in online systems to attribute detected fraud to these profiles. The gold standard fraudster profile collection (FPC) module described in the previous section performs task (1). In the following, we focus on task (2), which we break into two sub-problems:

- **Fraud-Component Detection Problem**. Given a product $A \in \mathcal{A}$, return a set of components $C_A = \{C_1, .., C_k\}$, where any $C_{j=1..k}$ consists of a subset of the user accounts who posted an activity for A, s.t., those accounts are either controlled by a single worker in $\mathcal{W}$, or are honest.
- **Component Attribution Problem**. Given $\mathcal{W}$ and a component $C \in C_A$, return the identity of the worker in $\mathcal{W}$ who controls all the accounts in the component, or $\perp$ if the accounts are not controlled by a worker.

Algorithm 3 MCDense: Min-Cut based Dense component detection. We set η to 5 and tau to 0.5.

Input: G = $(\mathcal{U}, \mathcal{E}_w)$: input graph
n := $|\mathcal{U}|$
Output: $\mathcal{C} := \emptyset$: set of node components

```
1.  MCDense(G){
2.      if (nodeCount(G) < η) return;
3.      (G1, G2) := weightMinCut(G);
4.      if ((ρ(G1) > ρ(G) & ρ(G2) > ρ(G))
            & (ρ(G) < τ)){
5.          MCDense(G1); MCDense(G2);
6.      else
7.          C := C ∪ G;
8.          return;
9.      end if
```

Dolos's FCD and CA modules respectively, provide solutions to these sub-problems. In the following, we detail the FCD and CA modules.

6.5.2 Fraud Component Detection (FCD) Module

In order to identify communities, each controlled by a different fraudster, we leverage the adversary Trait 4, that the accounts controlled by one fraudster are likely to have reviewed significantly more products in common than with the accounts controlled by another fraudster. We introduce *MCDense*, an algorithm that takes as input the **co-activity graph** of a product A, and outputs its *fraud components*, sets of user accounts, each potentially controlled by a different worker. We define the **co-activity graph** of a product A as $G = (\mathcal{U}, \mathcal{E}_w)$, with a node for each user account that posted an activity for A (see Figure 6.9 for an illustration). Two nodes $u_i, u_j \in \mathcal{U}$ are connected by a weighted edge $e(u_i, u_j, w_{ij}) \in \mathcal{E}^w$, where the weight w_{ij} is the number of products on which u_i and u_j posted activities in common.

MCDense, see Algorithm 3, detects densely connected subgraphs, each subgraph being minimally connected to the other subgraphs. Given a graph $G = (\mathcal{U}, \mathcal{E}_w)$, its triangle density is $\rho(G) = \frac{t(V)}{\binom{|V|}{3}}$, where $t(V)$ is the number of triangles formed by the edges in $\mathcal{E}^w$. This definition differs from the one used by Tsourakakis [Tso15] in the DSG algorithm (see § 6.8.4). Thus, unlike ρ_D that can be larger than 1, $\rho \in [0, 1]$.

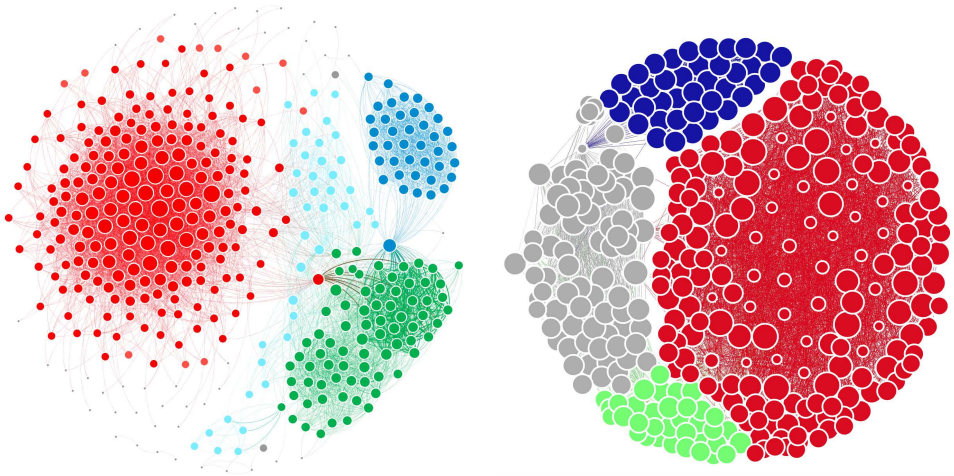
MCDense recursively divides the co-activity graph into two minimally connected subgraphs: the sum of the weights of the edges crossing the two subgraphs, is minimized. If both subgraphs are more densely connected than the initial graph (line 4) and the density of the initial graph is below a threshold τ, MCDense treats each subgraph as being controlled by different workers: it calls itself recursively for each subgraph (lines 5,6). Otherwise, MCDense considers the undivided graph to be controlled by a single worker, and adds it to the set of identified components (line 8).

We have used the gold standard set of accounts controlled by the 23 fraudsters detailed in the previous section, to empirically set the τ threshold to 0.5, as the lowest density of the 23 groups of accounts revealed by the fraudsters was just above 0.5.

MCDense converges and has $O(|\mathcal{E}_w||\mathcal{U}|^3)$ complexity. To see that this is the case, we observe that at each step, MCDense either stops or, at the worst, "shaves" one node from G. The complexity follows then based on Karger's min-cut algorithm complexity [Kar93].

6.5.3 Component Attribution (CA) Module

Given a set of fraud worker profiles $\mathcal{FW}$ and a set of fraud components returned by the FCD module for a product A, the component attribution module identifies the workers likely to control the accounts in each component. To achieve this, DOLOS leverages the unique writing style of human fraudsters to fuse elements from computational



(a) (b)

Figure 6.9: Co-review graph of user accounts reviewing a popular horoscope app in Google Play (name hidden for privacy). Nodes are accounts. 4 Upwork workers each revealed to control the accounts of the same color. Two accounts are connected if they post reviews for the same apps. Node sizes are a function of the account connectivity. (b) For the same app, DOLOS found these 4 tightly connected groups of accounts, and correctly attributed 3 groups to the fraudsters controlling them.

linguistics, e.g., [OCCH11, LLK+11], and author de-anonymization, e.g., [OG16]. Specifically, we propose the following 2-step component attribution process:

CA Training. Identify the products reviewed by the accounts controlled by each fraudster $W \in \mathcal{FW}$. For each such product, create a *review instance* that consist of all the reviews written by the accounts controlled by W for A. Thus, each review instance contains only (but all) the reviews written from the accounts controlled by a single fraudster, for a single product. Extract stylometry features from each review instance of each fraudster, including character count, average number of characters per word, and frequencies of letters, uppercase letters, special characters, punctuation marks, digits, numbers, top letter digrams, trigrams, part of speech (POS) tags, POS digrams, POS trigrams, word digrams, word trigrams and of misspelled words. Train a supervised learning algorithm on these features, that associates the feature values of each review instance to the fraudster who created it.

Attribution. Let $\mathcal{C}$ denote the set of components returned by MCDense for a product

Algorithm 4 DOLOS pseudocode. Given a set of crowdsourcing sites and peer-review site products, identify prolific fraudsters, accounts they control and targeted search rank fraud apps.

```
Input: Prod[ ] Products : monitored products
       site[ ] crowdSites : monitored sites
       int φ : threshold account number signal expertise
Output: < F, Acc >[ ] fraudsters : detected fraud
        Prod[ ] fraudProd : detected fraud products
1. Dolos (){
2.  while (true) do
3.      ⟨ F, Acc ⟩[ ] fraud := FPC.getSeeds(crowdSites);
4.      candidates.add(fraud); CA.train(candidates);
5.      for each prod in Products do
6.          C := MCDense.getComponents(prod);
7.          if (C.size ≠ 0) then fraudProd.add(prod);
8.          for each c in C do
9.              F f := CA.attribute(c, candidates);
10.             UserAcc a := candidates.getAccounts(f);
11.             a.add(c.accounts);
12.     for each ⟨ f, a ⟩ in candidates do
13.         if (a.size ≥ φ) then
14.             fraudsters.add(⟨ f, a ⟩);
15.             candidates.remove(⟨ f, a ⟩); }
```

A. For each component $C \in \mathcal{C}$, group all the reviews written by the accounts in C for product A, into a review instance, r. Extract r's stylometry features and use the trained classifier to determine the probability that r was authored by each of the fraudsters in $\mathcal{FW}$. Output the identity of the fraudster with the highest probability of having authored r.

6.5.4 Putting It All Together

Algorithm 4 shows the pseudocode of DOLOS. DOLOS takes as input a list of crowdsourcing sites and a list of products, and generates a list of identified prolific fraudsters and accounts that they are suspected to control in the online service, along with a

list of the products on which they have performed search rank fraud.

DOLOS uses FPC (see Section 6.4.3) to identify a fresh set of seed fraud from crowdsourcing sites, that consists of a new set of crowdsourcing site workers F, along with a set of user accounts Acc that each worker controls in the peer-review site (Algorithm 4, line 3). It then adds this seed information to the set of candidate fraud workers and uses it to re-train the component attribution (CA) module (line 4).

For each product received in its input (line 5), DOLOS uses MCDense to find the densely connected components of its co-review graph (line 6). If it finds at least one such component, it adds the product to the list of products targeted by search rank fraud (line 7), then, for each component, it uses the trained CA module to attribute the accounts in the component (line 9), and adds the accounts to the list of accounts controlled by the identified fraud worker (lines 10-11). At the end of this process, DOLOS plucks the expert fraud workers (i.e., who now control more than the threshold ϕ user accounts) and adds them to the list of fraudsters that it outputs (lines 12-15).

DOLOS repeats the above steps each time FPC identifies more seed ground truth data (line 2).

6.6 Fraud De-Anonymization Oracles

We leverage the observation that fraud workers know the user accounts that they control, to introduce a novel approach to validate fraud de-anonymization solutions. The protocol consists of 2 main interaction steps. In the first step, we ask each participant, i.e., recruited human fraud worker, to reveal m user accounts that they control in Google Play, by sending their Google e-mail addresses associated with these accounts. We then use a depth-2 breath first search approach to collect (1) all the apps reviewed by the m accounts and (2) all the reviewers of these apps. We apply

Figure 6.10: Anonymized screenshots of 3 questionnaire pages, for accounts (left) revealed in step 1 to be controlled by the participant, (center) known not to be controlled, and (right) suspected by our fraud detection module to be controlled by the participant.

a fraud de-anonymization solution to identify n new, *candidate accounts*, i.e., other Google Play accounts suspected to be controlled by the same participant.

For the second interaction step, we have designed a questionnaire that asks the participant to confirm if they control each of these n candidate accounts, see Figure 6.10. Specifically, for each account, we show the account's profile photo and name, and ask the participant if they control the account. We provide 3 options, "Yes", "No" and "I don't remember".

Participant validation. We have developed the following tests to validate participant attention and honesty:

- **Attention check**. In addition to the n candidate accounts, we add to the questionnaire q other *test* accounts, for which we know the answer: (1) accounts that we know that the participant controls, i.e., picked randomly from among the m accounts revealed in the first step, and (2) accounts that we know that the participant does not control, i.e., accounts that have at least 20 followers and significant other activities in Google Plus (posting photos, videos). We present the questions for the $n + q$ candidate and test accounts, in randomized order.

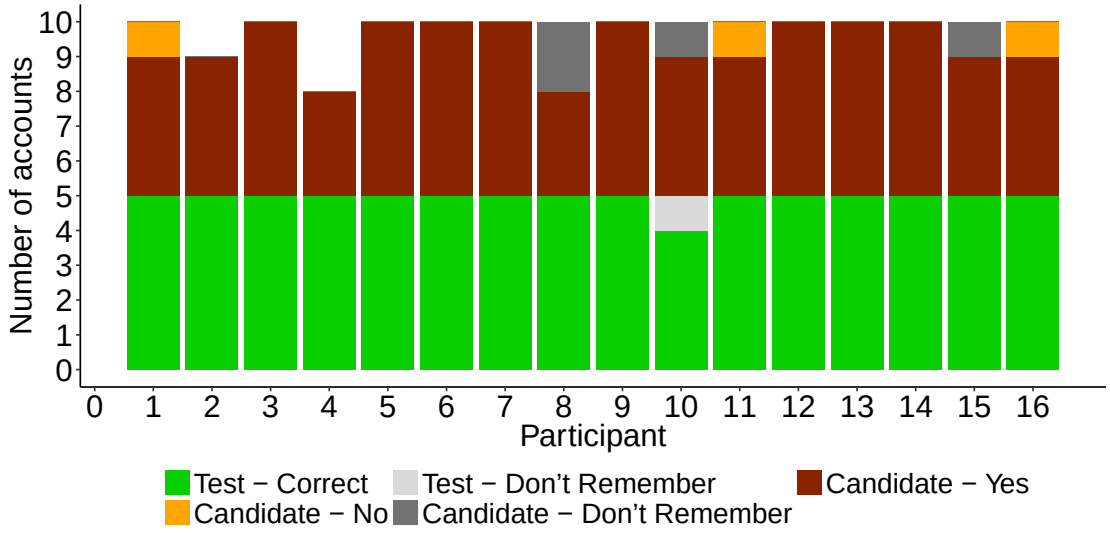


Figure 6.11: Results of data validated by 16 human fraud worker participants. We achieved an overall precision of 91%.

• **E-mail knowledge**. Each Google Play account A has an associated e-mail address E. Given E, one can easily retrieve the account A. However, E is not public, and, given only knowledge of A, one cannot find E. We leverage this observation to ask each participant to reveal the e-mail address E of each Google Play account A that they claim to control. We use E to find the corresponding account A'. The participant fails this test if A' does not exist or $A' \neq A$.

• **E-mail based validation**. To verify ownership of claimed accounts, we send the questionnaire to one of the m e-mail addresses revealed in the first step (randomly chosen).

• **Token and e-mail based validation**. To verify ownership of accounts confirmed in the questionnaire, we choose randomly one of the n accounts confirmed, and send to its corresponding e-mail address, a random, 6 character token. The accounts verify iff. the participant can reproduce the token.

6.7 User Study

We have recruited 16 fraud workers from India (4), Bangladesh (4), UK (2), Egypt (2), USA (1), Pakistan (1), Indonesia (1), and Morocco (1), 12 male and 4 female, who claimed to control between 40 to 500 accounts (M=211, SD=166). We have used these participants to evaluate the performance of our algorithm. We have set m=10, n=5 and q=5, thus each participant reveals 10 accounts controlled in Google Play, then further confirms or denies control of 5 other detected accounts, and 5 test accounts. We have used the 10 accounts revealed by each participant in the first step, to collect (via BFS) 718 apps, 265,724 reviewers and 341,993 reviews in total. We collected up to 175 apps, 37,056 reviews and 22,848 reviewers from a single worker. The participation incentive was set to $10 for each participant.

Ethical considerations. We have developed IRB-approved protocols to ethically interact with participants and collect data. We have not asked the participants to post any fraud on the online service. We restricted the volatile handling of emails and photos of accounts revealed by participants, to the validation process. We have immediately discarded them after validation. We believe that this information cannot be used to personally identify fraudsters: recruited fraudsters control between 40-500 accounts each (M=211, SD=166) thus any such account is unlikely to contain PII. Further, since we do not preserve these emails and photos, their handling does not fall within the PII definition of NIST SP 800-122. Under GDPR, the use of emails and photos without context, e.g., name or personal identification number, is not considered to be "personal information".

6.7.1 Results

Figure 6.11 shows that 15 of the 16 participants have provided correct responses to all 5 test accounts. The remaining participant answered "I don't remember" for a single

test account, known not to be controlled by the participant. We have thus decided to keep the data from all participants. Further, for participants 2 and 4, we found less than 5 suspected accounts (i.e., 4 and 3 respectively).

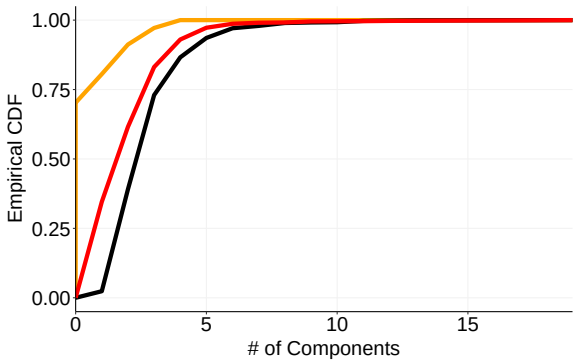
We observe that 10 out of 16 participants have confirmed control (and passed our verification) of all proposed accounts. 5 participants confirmed control of 4 out of 5 recommended accounts and 1 participant confirmed control of only 3 accounts out of 5 recommended accounts. Our algorithm's precision ($\frac{TP}{TP+FP}$, where TP is the number of true positives and FP is the number of false positives) is thus 91%, i.e., 7 unconfirmed accounts among 77 predicted. We note that for 3 out of the 7 unconfirmed accounts, the participants did not remember if they control them or not.

6.8 Empirical Evaluation

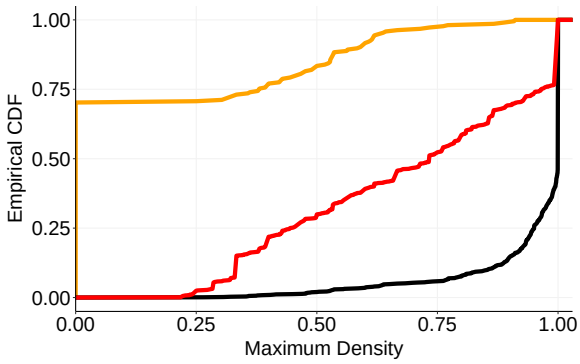
In this section we compare the results of DOLOS on fraud and honest apps, evaluate its de-anonymization accuracy, and present its results on 13,087 apps. Further, we compare MCDense with adapted dense sub-graph detection and loopy belief propagation solutions.

6.8.1 Fraud vs. Honest Apps

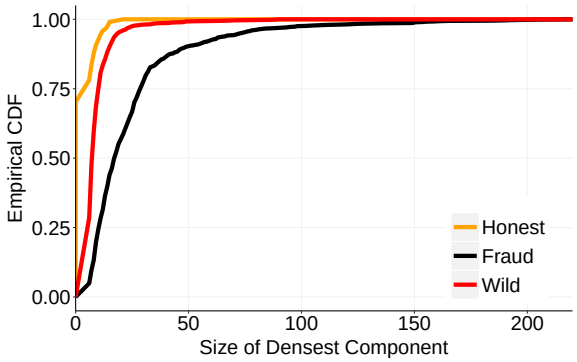
We evaluate the ability of DOLOS to discern differences between fraudulent and honest apps. For this, we have first selected 925 candidate apps from the longitudinal app set, that have been developed by Google designated "top developers". We have filtered the apps flagged by VirusTotal. We have manually investigated the remaining apps, and selected a set of 219 apps that (i) have more than 10 reviews and (ii) were developed by reputable media outlets (e.g., Google, PBS, Yahoo, Expedia, NBC) or have an associated business model (e.g., fitness trackers). We have collected 38,224 reviews and their associate user accounts from these apps.



(a)



(b)



(c)

Figure 6.12: MCDense: Cumulative distribution function (CDF) over 640 fraud, 219 honest, and 1,056 suspicious "wild" apps, of per-app (a) number of components of at least 5 accounts, (b) maximum density of an identified component and (c) size of densest component. (a) MCDense found at least 1 dense component in all the fraud apps. 70% of the honest apps had no component. (b) 94.4% of the fraud apps have a component with density exceeding 75%. Only 30% of the honest apps have a cluster with density larger than 0. 231 (21.87%) of the suspicious wild apps have at least 1 component with density 1. (c) 80% of the fraud apps vs. only 7% of the honest apps, have a densest component with more than 10 nodes. The largest densest component of a fraud app had 220 accounts, of a wild app had 90 accounts, and of an honest app had 21 accounts. We observe significant differences between fraud and honest apps.

Figure 6.12(a) compares the CDF of the number of components (of at least 5 accounts) found by MCDense per each of the 640 fraud apps vs. the 219 honest apps. MCDense found that all the fraud apps had at least 1 component, however, 70% of the honest apps had no component. The maximum number of components found for fraud apps is 19 vs. 4 for honest apps. Figure 6.12(b) compares the CDF of the maximum edge density (ratio of number of edges to maximum number of edges possible) of a component identified by MCDense per fraud vs. honest apps. 94.4% of fraud apps have density more than 75% while only 30% of the honest apps have a cluster with density larger than 0. The increase is slow, with 90% of the honest apps having clusters with density of 60% or below. Figure 6.12(c) compares the CDF of the size of the per-app densest component found for fraud vs. honest apps. 80% of the fraud apps vs. only 7% of the honest apps, have a densest component with more than 10 nodes. The largest, densest component has 220 accounts for a fraud app, and 21 accounts for an honest app. We have manually analyzed the largest, densest components found by MCDense for the honest apps and found that they occur for users who review popular apps such as the Google, Yahoo or Facebook clients, and users who share interests in, e.g., social apps or games.

6.8.2 De-Anonymization Performance

We have implemented the CA module using a combination of JStylo [JSt] and supervised learning algorithms. We have collected the 111,714 reviews posted from the 2,664 attributed, fraudster controlled user accounts of § 6.4.3. The reviews were posted for 2,175 apps. We have grouped these reviews into instances, and we have filtered out those with less than 5 reviews. The remaining total is 6,046 instances, 40 to 1,664 per fraudster. Figure 6.13(a) shows their distribution among the 23 fraudsters who authored them.

Algo	Top 1	(TPR)	Top 3	Top 5
k-NN (IBK)	**1608**	(**95.0%**)	**1645**	1646
RF (Random Forest)	1487	(87.9%)	1625	**1673**
DT (Decision Tree)	1126	(66.5%)	1391	1455
SVM	1101	65%	1195	1214
NB (Naive Bayes)	569	36.5%	874	1067
SMO	1117	68.3%	1434	1548

Table 6.2: DOLOS attribution performance for the 1,690 instances of the 640 fraud apps. k-NN achieved the best performance: It correctly identifies the workers responsible for 95% (1608) of the instances. The Random Forest (RF) classifier however identifies the correct worker among the top 5 most likely authors, in 98.9% of the instances.

We have evaluated the performance of DOLOS (MCDense + CA) using a leave-one-out cross validation process over the 640 fraud apps (and their 1,690 review instances). Specifically, for each app A, CA trains a supervised learning algorithm on all the review instances minus the instances that were written for A. DOLOS then converts each fraud component returned by MCDense for A into a review instance, that contains the reviews written by its accounts for A. It then extracts stylometric features of this instance with JStylo [JSt], then uses the trained CA to determine the workers most likely to have authored it. Thus, DOLOS trains a different classifier for each test app.

We have used several supervised learning algorithms, including k-nearest neighbors (k-NN), Random Forest (RF), Decision Trees (DT), Naive Bayes (NB), Support Vector Machine (SVM), and Sequential Minimal Optimization (SMO).

Instance level performance. Table 6.2 shows the number of instances correctly attributed by DOLOS (out of the 1,690 instances of the 640 fraud apps) and corresponding true positive rate, as well as the number of instances where the correct worker is among DOLOS' top 3 and top 5 options. k-NN achieved the best performance, correctly identifying the workers responsible for posting 95% of the instances. We observe that k-NN correctly predicts the authors of 95% of the instances. Fig-

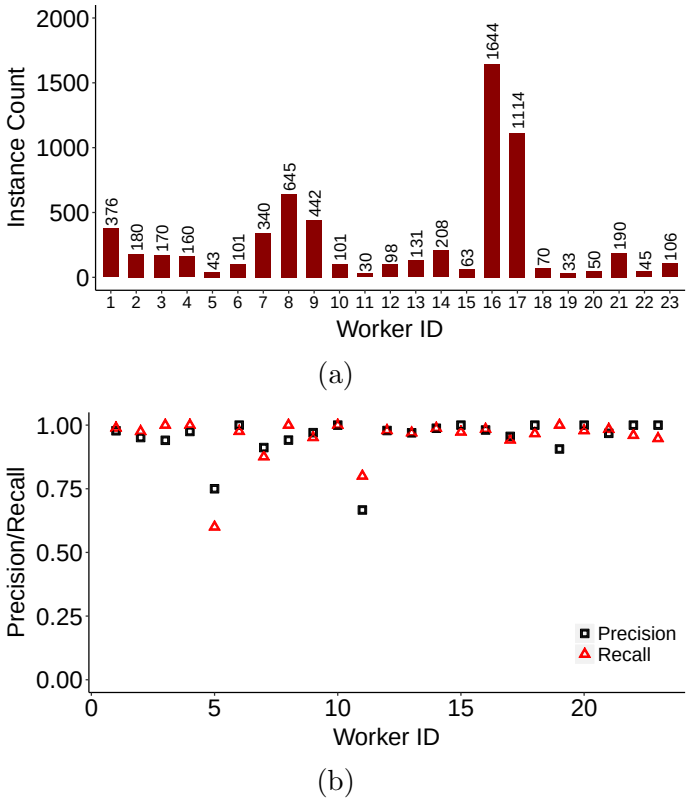


Figure 6.13: (a) Number of review instances collected from each of the 23 fraudster. Each review instance has at least 5 reviews, written by the accounts controlled by a single fraudster, for a single app. (b) DOLOS per-worker attribution precision and recall, over the 1,690 review instances of 640 fraud apps, exceed 87% for 21 out of the 23 fraudsters.

Algo	1 worker	50%-recall	70%-recall	90%-recall
RF	624	622	537	465
SVM	574	517	325	284
k-NN	**625**	**625**	**585**	**557**
DT	554	510	315	264
NB	379	256	128	125
SMO	533	492	318	265

Table 6.3: DOLOS app level recall: the number of apps for which DOLOS has a recall value of at least 50%, 70% and 90%. k-NN identifies at least one worker for 97.5% of the 640 fraud apps, and 90% of the workers of each of 557 (87%) of the apps.

Algo	50%-prec	70%-prec	90%-prec
RF	573	434	359
SVM	460	249	209
k-NN	**578**	**483**	**444**
DT	446	260	208
NB	217	100	99
SMO	436	264	212

Table 6.4: App level precision: the number of apps where its precision is at least 50%, 70% and 90%. The precision of DOLOS when using k-NN exceeds 90% for 69% of the fraud apps.

ure 6.13(b) zooms into per-fraudster precision and recall, showing the ability of DOLOS to identify the instances and only the instances of each of the 23 workers. For 21 out of 23 workers, the DOLOS precision and recall both exceed 87%.

App level performance. Table 6.3 shows that when using k-NN, DOLOS correctly identified at least 1 worker per app, for 97.5% of the fraud apps, and identified at least 90% of the workers in each of 87% of the fraud apps. Table 6.4 shows that the precision of DOLOS in identifying an app's workers exceeds 90% for 69% of the apps.

Developer tailored search rank fraud. Upon closer inspection of the DOLOS identified clusters, we found numerous cases of clusters consisting of user accounts who reviewed almost exclusively apps created by a single developer. We conjecture that those user accounts were created with the specific goal to review the apps of the developer, e.g., by the developer or their employees.

6.8.3 Dolos in the Wild

To understand how Dolos will perform in real life, we have randomly selected 13,087 apps from Google Play, developed by 9,430 distinct developers. We monitored these apps over more than 6 months, and recorded their changes once every 2 days. This enabled us to collect up to 7,688 reviews per app, exceeding Google's one shot limit of 4,000 reviews. We collected the data of the 586,381 distinct reviewers of these apps, and built their co-activity graphs.

MCDense found at least 1 dense component of at least 5 accounts in 1,056 of the 13,087 apps (8%). Figure 6.12 compares the results of MCDense on the 1,056 apps, with those for the fraud and honest apps. The CDF of the number of components found by MCDense for these "wild" apps is closer to that of the fraud apps than to the honest apps: up to 19 components per app, see Figure 6.12(a). The CDF of the maximum density of per app components reveals that 231 of the 1,056 apps (or 21.87%) had at least 1 component with edge density 1 (complete sub-graphs). The CDF of the size of the densest components (Figure 6.12(c)) found per each of the wild apps shows that similar to the 640 fraud apps, few of these apps have only 0 size densest components. The largest component found by MCDense for these apps has 90 accounts.

Validation of fraud suspicions. Upon close inspection of the 231 apps that had at least 1 component with edge density of 1 (i.e., clique), we found the following further evidence of suspicious fraud being perpetrated. (1) **Targeted by known fraudsters**: 169 of the 231 apps had received reviews from the 23 known fraudsters (§ 6.4.3). One app had received reviews from 10 of the fraudsters. (2) **Review duplicates**: 223 out of the 231 apps have received 10,563 *duplicate* reviews (that replicate the text of reviews posted for the same app, from a different account), or 25.55% of their total 41,339 reviews. One app alone has 1,274 duplicate reviews, out

Algorithm 5 DSG: Densest Sub-Graph algorithm.

Input: G = (V, E): input graph
n := $|V|$
Output: SG: optimum subgraph
1. Graph H := G;
2. double r := ρ_D(H); # holds max density
3. Graph SG := G
4. **for** i := 2 **to** n **do**
5. v := least connected node of H;
6. H := H - {v}
7. **if** (ρ_D(H) > r) **then**
8. SG := H;
9. r := ρ_D(H);
10. **end if**
11. **end for**
12. return SG;

of a total of 4,251 reviews. (3) **Fraud re-posters**: our longitudinal monitoring of apps enabled us to detect fraud re-posters, accounts who re-post their reviews, hours to days after Google Play filters them out. One of the 231 apps received 37 fraud re-posts, from the same user account.

6.8.4 MCDense Evaluation

MCDense Competitors

We adapt two existing solutions to the fraud-component detection problem and compare them against MCDense.

DSG: Adapted Densest SubGraph approach. We first compare MCDense against DSG, a densest subgraph approach that we adapt based on [Tso15]. DSG, whose pseudocode is shown in Algorithm 5, iteratively identifies multiple dense subgraphs of an app's co-activity graph $G = (U, E)$, each suspected to belong to a different worker. DSG peels off nodes of G until it runs out of nodes (lines 4-11).

During each "peeling" step, it removes the node that is least connected to the other nodes (lines 5-6). After removing the node, the algorithm computes and saves the density of the resulting subgraph (lines 7-10). The algorithm returns the subgraph with the highest density. We use the *triangle density* definition proposed in [Tso15], $\rho_D = \frac{t(U)}{|U|}$, where $t(U)$ is the number of triangles formed by the vertices in U. DSG uses this greedy strategy iteratively: once it finds the densest subgraph D of G, DSG repeats the process, to find the densest subgraph in $G - D$. The nodes in each identified densest subgraph are well connected among themselves, but not well connected to the nodes in the previously identified subgraphs.

LBP: Adapted Loopy Belief Propagation approach. We adapt a Loopy Belief Propagation approach [ACF13a] to formulate the problem of detecting fraudulent user accounts as a network classification task on $\mathcal{G}$. The resulting algorithm, LBP, assigns labels to the user account nodes of the co-activity graph. Specifically, the graph is modeled as a pairwise Markov Random Field (MRF) [YFW03], where each user account node has a random variable Y_i that can take values from the user class domain $\mathcal{L} = \{honest, fraud\}$ (i.e., the label space), encoding the belief that the node is fraudulent.

In MRFs, the memoryless Markov property implies that in the undirected network, the label of a node only depends on its neighbors. Then, the overall joint probability distribution is written as the normalized product of factors associated with the nodes and edges [KF09]: $\mathcal{P}(\mathbf{y}) = \frac{1}{Z} \prod_{Y_i \in \mathcal{U}} \phi_i(y_i) \prod_{(Y_i, Y_j, w_{ij}) \in \mathcal{E}_w} \psi_{ij}^w(y_i, y_j)$, where $\mathbf{y}$ represents an assignment of labels to each of the nodes in $\mathcal{U}$, and Z is the normalization constant. $\phi : \mathcal{L} \to \mathbb{R}^+$ represents the "prior probability" for each node, and $\psi^w : \mathcal{L} \times \mathcal{L} \to \mathbb{R}^+$ are the "compatibility potentials".

We adapt the priors ϕ_i and compatibility potentials ψ_{ij} to capture the behavioral dynamics of the users in the Google Play review ecosystem. We have experimented with two types of priors. First, the priors are all 1/2, modeling the lack of knowledge

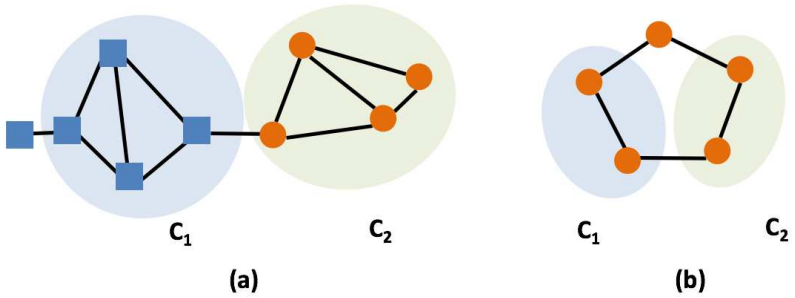


Figure 6.14: Illustration of p-coverage and p-SCC scores that measure the quality of detected community partitions (shown as ovals). (a)Graph with two workers: one controls the square nodes and one the round nodes. The partition provides (50%,100%)-coverage and (50%,100%)-SCC. (b) Graph with a single worker (circles). p-coverage $\neq$ p-SCC: the partition provides (100%,80%)-coverage but only (0%,80%)-SCC.

on the status of nodes/accounts. Second, we chose the prior of a user node u_i having label *honest* to be inversely proportional to the average weight of u_i's edges, i.e., $\phi_i(honest) = \frac{|N(u_i)|}{\sum_{j \in N(u_i)} w_{ij}}$; $\phi_i(fraud) = 1 - \phi_i(honest)$. The results shown are over the latter approach, which proved to be more effective.

Further, the compatibility potentials $\psi_{ij}(y_i, y_j)$, capture the likelihood of a node u_i with assigned label y_i to be neighbor of a node u_j with label y_j, when u_i and u_j have a link of weight w between them. We defined $\psi_{ij}(y_i, y_j)$ as follows. If u_i is honest, $\psi_{ij}(y_i, y_j)$ is independent of w_{ij}. However, when u_i is "fraud" $\psi_{ij}(y_i, y_j)$ depends on w_{ij}: $\psi_{ij}(y_i = \text{“}fraud\text{”}, y_j = \text{“}honest\text{”}) = \delta^{\log w_{ij}}$ decreases exponentially with w_{ij}, while $\psi_{ij}(y_i = \text{“}fraud\text{”}, y_j = \text{“}fraud\text{”}) = 1 - \delta^{\log w_{ij}}$ increases, where δ in $(0, 1)$.

Evaluation Scores

To evaluate MCDense, we introduce two coverage scores. Let $\mathcal{RA}(A) = \{a_1, .., a_k\}$ denote the set of user accounts who reviewed product A. Given the set $S = \{W_1, .., W_w\}$ of workers who wrote fake reviews for A, let $\overline{W_i} = \{a_{i_1}, .., a_{i_j}\}$ denote the accounts in $\mathcal{RA}(A)$ that are controlled by worker W_i, $i = [w]$. Then, a *partition* of $\mathcal{RA}(A)$ is a set of sets $\{P_1, .., P_p\}$, such that each account $a_i \in \mathcal{RA}(A)$, $i = [k]$, belongs to

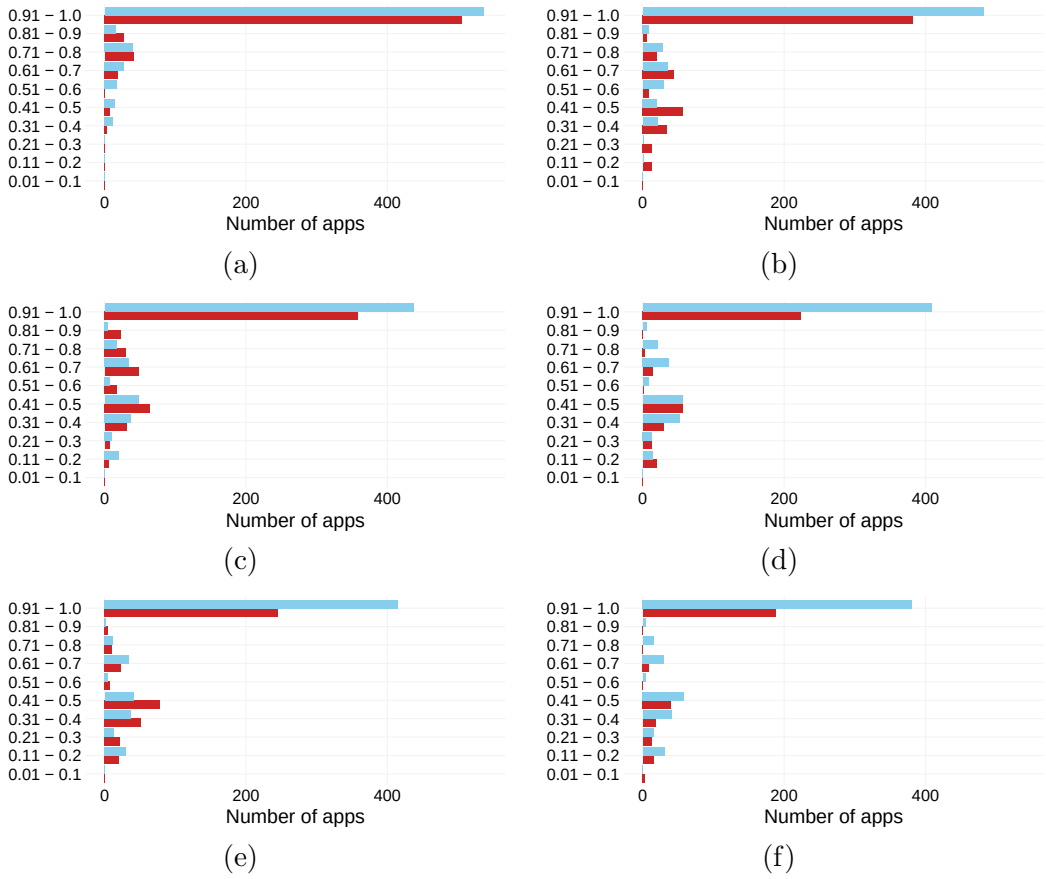


Figure 6.15: Comparison of MCDense and DSG distribution of coverage score and distribution of SCC score over 640 fraud apps. The y axis shows the p_1 value, and the x axis shows the number of apps for which MCDense and DSG achieve that p_1 value, when (a,b) $p_2 = 50\%$, (c,d) $p_2 = 80\%$, and (e,f) $p_2 = 90\%$. MCDense consistently outperforms DSG as it provides (a) (90%+, 50%)-coverage for 537 (83%) of the apps vs. DSG's 506 apps, and (b) (90%+, 50%)-SCC for 490 (75%) of the apps, vs DSG's only 383 apps, and (e) (90%+, 90%)-coverage for 415 (65%) of the apps vs. DSG's 245 apps, and (f) (90%+, 90%)-SCC for 381 of the apps, vs. DSG's 188 apps, which is half of MCDense.

exactly one of these sets. Let $H = \mathcal{RA}(A) \setminus \cup_{i=[w]} \overline{W_i}$, be A's "honest" reviewers, i.e., accounts not known to be controlled by a worker. The set $\{\overline{W_1}, .., \overline{W_w}, H\}$, forms a partition of $\mathcal{RA}(A)$.

Let $\mathcal{C} = \{C_1, .., C_c, H_C\}$ be the partition of $\mathcal{RA}(A)$ of the user accounts who reviewed an app A, returned by a fraud-component detection algorithm: $\forall a_i, a_j \in C_l$, are considered to be controlled by the same worker, and H_C is the set of accounts

considered to be honest. To quantify how well the partition $\mathcal{C}$ has detected the worker accounts $\overline{W_1}, .., \overline{W_w}$ who targeted A, we propose the *coverage* measure of worker $W_i \in S$ as follows:

Definition 6.8.1 *(Coverage) The coverage of worker $W_i \in S$ by a partition C is $cov_i(C) = \frac{|\overline{W_i} \cap (C_1 \cup .. \cup C_c)|}{|\overline{W_i}|}$. Given $p \in [0,1]$, we say that W_i is "p-covered" by C if $cov_i(C) \geq p$. Then, we say that partition C provides a (p_1, p_2)-coverage of the worker set S, if p_1 percent of the workers in S are p_2-covered by C.*

Further, we introduce the *single component coverage* (SCC) of a worker $W_i \in S$ by a partition $\mathcal{C}$:

Definition 6.8.2 *(Single Component Coverage - SCC) The single component coverage of a worker $W_i \in S$ by a partition $\mathcal{C} = \{C_1, .., C_c\}$ is $SCC_i(C) = max_{j=[c]} \frac{|\overline{W_i} \cap C_j|}{|\overline{W_i}|}$. Given $p \in [0,1]$, we say that W_i is "p-single component covered" (or p-SCC) by C if $SCC_i(C) \geq p$. We say that partition C provides a (p_1, p_2)-SCC of the worker set S, if p_1 percent of the workers in S are p_2-SCC by C.*

p-SCC is about precision: a worker is p-single component covered by *Alg* only if at least p percent of its accounts belong to a single component discovered by *Alg*. In contrast, a worker is p-covered if p percent of its accounts belong to any component returned by *Alg*. As such, p-SCC will always be at most equal to p-coverage. Figure 6.14 illustrates the p-coverage and p-SCC measures on the co-activity graphs of two apps.

MCDense vs. DSG

Figure 6.15 compares MCDense and DSG in terms of their distributions of the p-coverage and p-SCC scores, over the 640 fraud apps. MCDense consistently outperforms DSG. For instance, Figure 6.15(a) shows that 537 apps are at least (90%+,

50%)-covered by MCDense, while only 507 apps achieve the same coverage for DSG. The difference is even higher for the p-SCC score: Figure 6.15(b) shows that 490 apps (75%) are at least (90%+, 50%)-SCC by MCDense, that is, at least 50% of the accounts controlled by 90% of the workers belong to *only one* of the components returned by MCDense. In contrast, only 383 apps are at least (90%+, 50%)-SCC by DSG.

The difference between MCDense and DSG becomes more pronounced as p_2 increases to 80% and 90%. For instance, Figure 6.15(c) shows that 438 apps are at least (90%+, 80%)-covered by MCDense, while only 359 apps achieve the same coverage for DSG. Figure 6.15(d) compares the p-SCC score: 409 apps (63%) are at least (90%+, 80%)-SCC by MCDense compared to only 225 apps that are at least (90%+, 80%)-SCC by DSG. Figure 6.15(e) shows that 415 apps are at least (90%+, 90%)-covered by MCDense, while only 245 apps achieve the same coverage for DSG. Figure 6.15(f) shows the p-SCC score: 381 apps (59%) are at least (90%+, 90%)-SCC by MCDense, but only less than half (188 apps) are at least (90%+, 90%)-SCC by DSG.

LBP Performance

Since LBP has no information (in the form of priors) about the accounts controlled by workers, we use it to determine which accounts are suspected of being controlled by *a* worker, as those whose final fraud belief exceeds 0.5. Figure 6.16 shows the box and whiskers plot of the precision and recall values of LBP, when δ ranges from 0.1 to 0.9. We observe that recall in this case is equivalent to our p-coverage score. In addition to precision and recall, we also use the notion of "prevalence": the ratio of the number of fraud labeled accounts to the total number of the app's accounts. This enables us to determine when LBP labels all the accounts as fraudulent. LBP

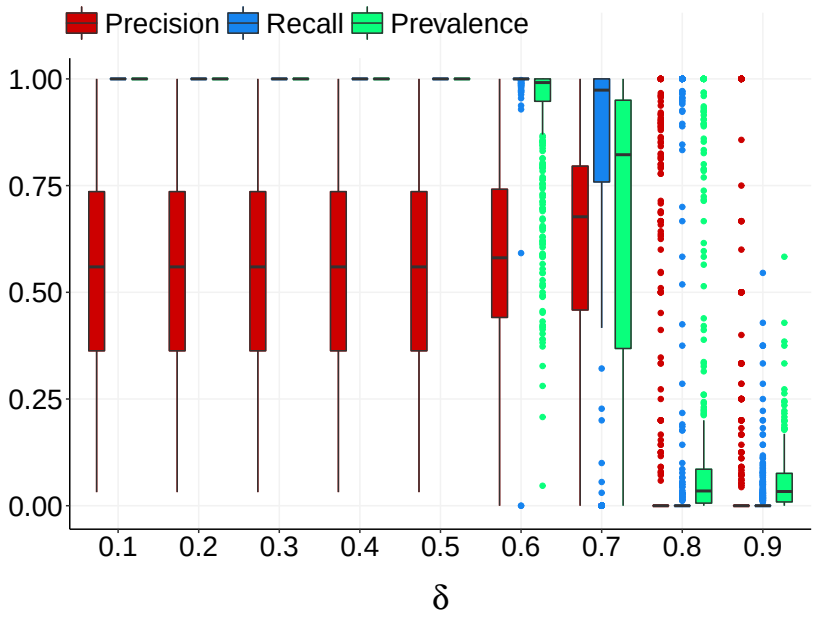


Figure 6.16: LBP identification of fraudulent accounts: precision, recall and prevalence, when $\delta = [0.1..0.9]$. LBP achieves best performance when $\delta = 0.7$, with a median precision of 69% and recall of 98%; the median prevalence of 82% shows that these values are not achieved by labeling all the accounts as fraudulent.

achieves the best performance when $\delta = 0.7$, with an average per-app recall exceeding 95%.

Thus, while LBP can be used to detect fake reviews, it cannot determine if all accounts detected as fraudulent are controlled by a single or multiple workers.

6.9 Research Contributions Acknowledgment

It is a pleasure to acknowledge the contributions of my co-authors; Nestor Hernandez, Dr. Duen Horng Chau and my supervisor Dr. Bogdan Carbunar for their roles in the teamwork. Thanks to Nestor for adapting and evaluating loopy believe propagation it on our ground truth data to the fraud de-anonymization problem. My special gratitude goes to Dr. Carbunar and Dr. Chau for providing critical feedback to shape the research, analysis, and manuscript.

6.10 Summary

We introduced the fraud de-anonymization problem for search rank fraud in online services. We have collected fraud data from crowdsourcing sites and the Google Play store, and we have performed user studies with crowdsourcing fraudsters. We have proposed DOLOS, a fraud de-anonymization system. DOLOS correctly attributed 95% of the fraud detected for 640 Google Play apps, and identified at least 90% of the workers who promoted each of 87% of these apps. DOLOS identified 1,056 out of 13,087 monitored Google Play apps, to have suspicious reviewer groups, and revealed a suite of observed fraud behaviors. DOLOS significantly outperforms adapted dense subgraph detection and loopy belief propagation competitors, in two coverage scores that we have developed.

CHAPTER 7

A Study of Crowdsourced Review Fraud

7.1 Introduction

Popular online services that provide millions of users with access to products, news, social relationships and peer-opinions, are besieged by cyberfraudsters, who skew public opinion and bias product reputation and popularity. In search rank fraud campaigns, cyberfraudsters leverage user accounts that they control on the online service, to fraudulently promote products e.g., through fake reviews, installs, and likes. Cyberfraudsters often act as guns-for-hire, i.e., connect with product developers through specialized [TSM16, RL16, AV16, AS16, AR16] or general crowdsourcing sites [Fiv, Upw, Fre].

Commercial sites employ proprietary solutions to detect and filter fraud [Cip16, Per16, MVLG13]. Similarly, a substantial body of academic research has focused on the detection aspect of the fraud problem. This work is built on assumptions about the adversarial behaviors and capabilities of search rank fraud workers, which are based on intuition, extracted from small datasets of fraud, or revealed by collaborators within commercial sites. This scarcity of ground truth knowledge of search rank fraud adversaries stems in part from the ethical and terms-of-service concerns associated with requiring recruited participants to post fraudulent information on existing products hosted on peer-opinion sites, e.g., [DCFJ+14, SWE+13, GHP+12, OCCH11].

In this paper we study the search rank fraud ecosystem that targets peer-opinion sites, in particular, Google Play. The diversity, responsiveness and revealed profitability of this ecosystem, suggest an inaccurate and incomplete understanding of the capabilities, behaviors and strategies of fraud workers, including their strategies to avoid detection.

Figure 7.1: Photo taken by fraud worker who participated in our study, with the premises and (anonymized) employees of his business. Photo reproduced with permission from the participant.

The goal of this paper is to determine which existing assumptions still hold, whether fraudsters have developed fraud detection strategies to avoid being identified by the peer-opinion site, to determine the capabilities, behaviors and strategies employed by fraudsters, and identify weakness that can be exploited by online systems and researchers, to identify and prevent fraud.

We survey adversarial assumptions made in academic research and introduce new hypotheses on capabilities, behaviors and strategies employed by search rank fraud adversaries. We develop questions that seek to evaluate the capabilities of search rank fraud workers and to confirm each of the previously identified assumptions, concerning e.g., fraud account life-cycle (e.g., creation, maintenance, closure), review creation and posting.

We first surveyed more than 80 top-tier research papers that study fraud and abuse in online systems and extracted key assumptions made when designing detection algorithms. We organize the resulting assumptions based on several categories: fraudster capabilities, review burstiness, review plagiarism, camouflage strategies,

collaboration, lockstep behaviors, and automation. Further, based on pilot studies (see below) and questions raised during interviews with fraudsters, we introduce several new high-level assumptions that concern credential and device usage, rehiring, account blending, and self-promotion.

We have developed a questionnaire that consists of more than 100 open ended questions, designed to investigate each of the existing and newly proposed assumptions. We have recruited 18 fraud workers from several sites (Facebook, Upwork, Fiverr, Zeerk and Peopleperhour) and conducted semi-structured interviews to collect answers to our questions. Figure 7.1 shows an anonymized snapshot of the employees of one of the recruited participants.

7.2 Background and Model

We consider an ecosystem that consists of the Google Play app market, and crowdsourcing sites. The app market hosts accounts for apps, developers and users. Developers use their accounts to host apps on the market, and upload information about them.

User accounts and validation: User accounts enable app market users to establish an online identity. Sites like Google increasingly pressure users into validating their accounts, e.g., by providing the calling number of a phone that they control. User then need to prove control of the device, e.g., by retrieving a code sent through SMS to that number.

App reviews. Users search for and install apps. Subsequently, they can post reviews for these apps, from any of her registered devices. A review consists of a star rating (1-5) and text, and also includes the profile photo and name of the user account from which it was posted. Users can log in multiple Google accounts from any one device. Users are then able to post reviews for an app, from all the accounts

logged in on the device where the app was installed, since the Google Play Store application allows them to switch back and forth between accounts. Therefore, a user can download an application once and review it as many times as the number of accounts she has logged in on the device.

In Google Play, some reviews are posted under "A Google User" while others under the account's name. Our experiments have shown that "A Google User" is someone who has a user id but has not activated his Google Plus account. Once the person activates her Google Plus profile, "A Google User" changes to the account's username the person used for her gmail account.

Search rank fraud and crowdsourcing. The search rank of Google Play apps has significant impact on the returns made by their developers. While the algorithm for computing the search rank of an app is kept secret, it is well known (e.g., [Ank13]) that it heavily depends on features such as the product's number of reviews and average rating. This knowledge has enabled a black market for *search rank fraud*. Specialized fraud workers (also referred to as fraudsters) connect with app developers to artificially boost the rank of their apps, by installing their apps, then posting fake reviews.

Fraudulent developers have incentives to maximize their subject's visibility via review manipulation for which they hire fraudsters. Thus, we also refer to developers as *employers*. Developers and fraudsters connect through a variety of sites, that range from general-purpose crowdsourcing sites [Fiv, Upw, Fre] and specialized fraud sites [TSM16, RL16, AV16, AS16, AR16], to social networks (e.g., Facebook groups). Developers either post app search optimization (ASO) jobs on which fraud workers can bid, or they recruit directly fraudsters who advertise on such sites.

7.3 Fraudster Assumptions

We first survey the fraudster assumptions made in more than 80 research papers. Second, we introduce several new high-level assumptions based on prior experience and pilot studies.

7.3.1 Survey of Previous Assumptions

We surveyed more than 80 top-tier research papers that study fraud and abuse in online systems and extracted key "previous assumptions" (PA) made when designing detection algorithms.

PA.1. Fraud Expertise, Organic Fraud: Crowdsourcing sites host workers who are search rank fraud experts, e.g. who control hundreds of user accounts, participate in many search rank fraud jobs, and are very active [YA15b, BXG+13, MKL+13a, SLK15, XZLW16, LCNK17, MLG12, XZ14, LFW+17a, FLCS15]. Conversely, product developers may also hire *organic fraud workers*, i.e., real, non-Sybil user account owners, via specialized crowdsourcing websites such as Microworkers or RapidWorkers [KCS18]. They may also hire "leader" workers who organize many such organic fraud workers to participate in the search rank fraud campaign [ZXL+18].

PA.2. Review Burstiness: Crowdsourced reviews are posted in a bursty fashion. Genuine reviews are likely to be posted over a reasonable long timeframe, whereas fake reviews are posted within a very short burst [FML+13a, HTS16, LFW+17a, HSB+15, BSLL+16a, XZLW16, Xu13, GGF14, BXG+13, KCA17, LNJ+10, MVLG13, MKL+13a, KCS18, LCNK17, YKA16, FLCS15, DCFJ+14, XWLY12].

PA.3. Lockstep Behaviors: Fraudsters synchronize the activities of their accounts:

They use their accounts to target the same product, at the same time, and in the same order [SMJ+15, TZX+15, YKA16, XZ15b, JCB+14, SLK15, XZ14, XZ15a, LFW+17a].

PA.4. Fraud Time Points: Product developers have specific points where they recruit fraudsters:

- **PA.4.1. Early Bird Fraud**: Fraudsters are hired early in the lifetime of a product, in order to be among its first reviewers and control its sentiment [FML+13a, YKA16, LNJ+10, MKL+13a, MLG12, Xu13, KM16].
- **PA.4.2. Proactive Fraud**: Developers hire fraudsters to proactively inject deceptive reviews and maintain a threshold average rating or popularity, or to reduce the impact of truthful feedback [KM16].

PA.5. Camouflage Strategies: Fraudsters employ a range of methods to avoid detection [ACF13a, HSB+16b, RA15b, PCWF07, RA15b, DCFJ+14]:

- **PA.5.1. Noisy Reviews**: Fraudsters also write genuine reviews, for products for which they have not been hired [ACF13a, FML+13a, WXLY11, KCS18, RA15b, DCFJ+14].
- **PA.5.2. IP Rotation**: Fraudsters change IP addresses frequently when posting reviews to thwart the review filter system of the online service [LCM+15, TMG+13].
- **PA.5.3. Emulators**: Fraudsters prefer to use mobile device emulators running on PCs instead of actual mobile devices, to post fake reviews [Xu13, LCM+15, SMJ+15].
- **PA.5.4. Hijacked Accounts**: Some fraudsters use hijacked accounts from honest users, and then the camouflage is indeed organic and they have realistic patterns of camouflage essentially similar to that of honest users [WXLY11, HSB+16b].
- **PA.5.5. Upvoting: Fraud and accomplice**: Fraudsters create two types of accounts, "fraud" and "accomplice". Fraud accounts are used to post the actual fraud, while the accomplice accounts boost the fraud's feedback rating [PCWF07].

PA.6. Review Writing: Previous work makes several assumptions about review writing strategies and techniques:

- **PA.6.1. Review plagiarism**: Fraudsters tend to reuse common linguistic patterns and copy reviews across similar products since they are not familiar with the item and crafting new reviews is time consuming [MKL+13a, FML+13a, Xu13, HTS16, LNJ+10, SE15, MVLG13, XZ15b, KCS18, LCNK17, RA15b, YA15b, MLG12, KCA17].
- **PA.6.2. Short Reviews**:

  Fake reviews are short [KCA17, MVLG13, JL07, KCS18, FML+13a, LCNK17].
- **PA.6.3. 1st person pronouns**: Crowdsourcing workers try to convey the impression that they have actually used the product by writing reviews in first person [KCA17, RA15b, KCLS17].

PA.7. Ratings: Previous work has made two assumptions about the ratings of fake reviews.

- **PA.7.1. Extreme Ratings**: Fraudsters are likely to give extreme ratings either to promote or demote a product (1 star or 5 star) [MKL+13a, ACF13a, KCA17, MVLG13, KCS18, RA15b, MLG12, XZ14, XZ15a].
- **PA.7.2. Rating Deviation**: Ratings by spammers often deviate from the average ratings given by other reviewers [MKL+13a, FML+13a, LCM+15, Xu13, WXLY11, HTS16, FFSG15, LNJ+10, MVLG13, WXLY12, XZ15b, XZ14, ZXGC13, RA15b, MLG12, XZ15a, KCS18].

PA.8. Collaboration: A fraudster may work with different sets of other fraudsters on multiple spamming tasks, causing groups sharing some common members [XZ15b, Xu13, DCFJ+14].

PA.9. Review Bots: Fraudsters use autonomous programs to manage multiple accounts at once [SMJ+15, DCFJ+14, YVC+17, DCFJ+14].

PA.10. Account Name Variability: Fraudsters employ limited variability in naming patterns for the accounts that they control [TMG+13].

PA.11. Singleton Accounts: Targeted products receive many fake reviews posted by singleton accounts, i.e., who only reviewed that product [MKL+13a, YKA16, RA15b, SE15, XWLY12].

7.3.2 New Assumptions

Based on pilot studies and questions raised during interviews with fraudsters, we introduce several new high-level assumptions that we list below.

NA.1. Fraud Team Organization: Fraudsters work in teams and organize to distribute tasks.

NA.2. User Account Validation: Fraud workers have developed techniques to validate their user accounts with only a few phone numbers.

NA.3. Mobile Device Types: To save money, fraudsters tend to have cheap, low-end devices.

NA.4. Devices vs. accounts: The number of devices used for a job is linear in the number of accounts controlled.

NA.5. Credential Reuse: Memorability affects fraudster choice of user account names and passwords.

NA.6. Account Blending: Fraudsters use a blend of older, previously used accounts with newly created accounts, in their jobs. This enables them to replenish or increase their base of accounts controlled, build the reputation of older accounts, and reduce chances of detection of lockstep behaviors and singleton use.

NA.7. Retention Installs: Fraud workers keep apps that they target, installed on their devices for a longer time interval, and do not delete them immediately after the job completion.

NA.8. Developer-based reviews: Review text can be provided by the developer.

NA.9. Self-promotion: Sybil accounts tend to upvote each other's reviews on the online system.

NA.10. Work Verification: Developers verify that recruited fraud workers satisfy the terms of the job.

NA.11. Fraudster Re-Hires: Developers re-hire successful fraudsters at later times, e.g., when search rank decreases.

7.4 Fraudster Survey

We have recruited 18 fraud workers from Facebook(9), Upwork(3), Fiverr(3), Zeerk(2), Peopleperhour(1). We launched the first survey in August 2018, and adopted an incremental payment strategy that consists in paying workers 5 USD for every 15 minutes of their time. Given the nature of our questions, we conjectured that a cooperating participant will easily take 45 minutes, and this payment structure is an incentive for cooperation

We advertised the survey as "Understanding Search Rank Optimization". We organized each interview to take place over Skype, to asses the fraudster diligence to the task and discourage would-be cheaters by increasing the effort to provide a legitimate response [OK14]. We have audio recorded the interviews and referred to them later for further analysis.

We converted the assumptions of Section 7.3 into open-ended questions. We do this because we do not want to influence the participants, the universe of possible answers is unknown, or there may be more possible answers than we can easily display, and also to measure qualitative aspects of a fraudster experience in ASO jobs [OK14, SBN96].

Considering the sensitivity of the topic in question, we have formulated several questions in third person to reduce social-desirability bias. For instance, we prefer "Do you know any developers/employers who ask freelancers to post reviews for recently launched apps?" to "Have you had a job requesting reviews for recently launched apps?". To avoid context switching, we grouped related questions so that participants can more easily and quickly retrieve related information from memory. Further, to minimize the effects of leading questions, we formulate them in a fully neutral way without examples or additional information [OK14]. We list our survey questions in Appendix 7.8. We have developed our protocols to interact with participants and collect data in an IRB-approved manner (Approval #: IRB-15-0219@FIU and IRB-18-0077@FIU).

Ethical considerations. The ability to massively link individual user accounts with the device types used to post reviews is a sensitive privacy breach. As a consequence, we have carefully handled all data collected according to GDPR and other privacy recommendations and good practices. Specifically, we have only used the link between the user account and the posted review to identify target reviews. With a list of fraudulent reviews, we have collected the device types used and never link them back to the user accounts that originated them. Once data was collected, we have deleted the link between the device types used and reviews and generated statistics that allowed us to validate our assumptions. We also note that we have not collected any other information about devices used to post reviews, expect their type.

7.5 Assumption Validation

In the following, we use the interview results and the validation data to evaluate the fraud assumptions and conjectures from § 7.3.

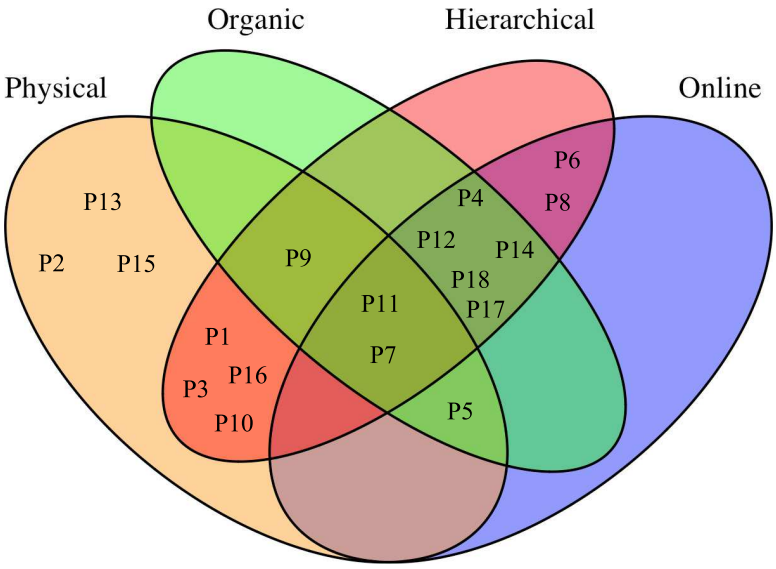


Figure 7.2: Venn diagrams showing multiple labels of the participants.

7.5.1 Fraud Types and Organizations

All the 18 participants claimed to be part of organizations dedicated to posting fraud, for a variety of peer-opinion services, including Google Play. We classify the self-reported teams into several categories, based on their location (i.e., physical, online), organization type (hierarchical vs. flat), and the type of fraud (organic vs. inorganic). Figure 7.2 shows the Venn diagram of the 18 participants grouped according to several of these categories.

Co-located vs. online. 6 participants claimed to be part of physically co-located teams, with 5 of them claiming to have brick and mortar offices. 10 participants were part of strictly online teams, which communicate through messaging apps such as Facebook and Whatsapp. 2 participants claimed to be part of hybrid teams, both with co-located and online team members.

Organization structure. The participants described diverse organizational structures:

- *Hierarchical organization.* 12 participants claimed a hierarchical structure of their organizations. 9 participants (P3, P4, P5, P6, P8, P9, P10, P12, P14, P17)

described specific roles in their organizations, that include *job managers*, who interface with the developers and manage work from the marketplace, and *team admins*, each organizing multiple *review posters*. Some of these organizations are hybrids. For instance, P14 is part of a team of 4 physically co-located members (no hierarchy), and they manage 30 online team members. P11 has his own devices and accounts, and also a "friends and family" (FnF) organization. For instance, P5 has a team of 12 and also a Facebook group with organic users. P7 has a team of 50 people but also runs ad-hoc campaigns to recruit more reviewers. Facebook group.

- *Organic fraud.* 6 participants (P4, P5, P6, P12, P17, P18) claimed to also organize or be part of online teams of "organic" users, i.e., fraudsters who use their personal accounts to post fraud. The hierarchy is implicit: the participants is the team lead, who gets jobs from employers, distributes them to his teams, then shares the profit. They recruit, organize and sync with organic users through social media, e.g., dedicated Facebook and Whatsapp groups, and microworker crowdsourcing sites) .
- *Flat organization.* 2 participants (P13, P15) claimed to work in teams where each member controls their own accounts and collaborate on jobs, but have no hierarchy.

Finding 1: PA.1 (Fraud types) is correct in that all the fraudster types assumed, are reported among our participants. However, our study also revealed the dangers of only considering one type of adversary. Since adversaries may be hubrid (e.g., both fraud posting experts and in control of an army of organic fraudsters), focusing on a single adversary type risks missing the other fraud.

Finding 2: C.1 (Fraud Organization) also validates, but our survey exceeds expectations in terms of variations in team organizations.

PA.3. Outside collaborations. Only 5 of the participants admitted to their teams collaborating with external fraudster teams. P2 said that his company collaborates with 4-5 other companies when developers request more reviews than they can post, and share the profit. 2 participants said that they only tried to collaborate once, with 1 claiming unpleasant results.

Profit sharing. Of the 9 participants who claimed profit sharing, 1 claimed to pay team members a monthly salary, 1 claimed an even split among members, 3 mentioned preferential cuts for the job manager (10-25%) and team lead (10-50%) and equal split of the rest among the actual review posters. 2 participants claimed a flat rate for the review posters ($0.40 per review).

7.5.2 Fraudster Capabilities

Figure 7.3 shows the claimed team sizes by each participant. 5 participants (P6, P11, P12, P17, P18) claimed to work individually but to also control online groups (e.g., on Whatsapp)

13 participants claimed to have a team with more than 10 members. Notably, P4 claimed to run a big company with around 150 people in their team, who organize 15,000 organic fraudsters through virtual (WhatsApp, Facebook) groups.

Accounts controlled. Figure 7.3 shows the number of user accounts claimed to be controlled by or accessible to each of the 18 participants. Most participants control a few hundred accounts, however, a few control or have access to several thousand user accounts. One participant, part of a team of 13 workers, claimed to control 80,000 user accounts, of which 47,000 were purchased.

Duration and scale of expertise. Participants claimed to have between 1 and 6 years of experience in ASO jobs (M = 3.03, SD = 1.53). They also claimed to have worked on between 150 and 4,000 apps in total, and between 6 and 50-60 apps in the

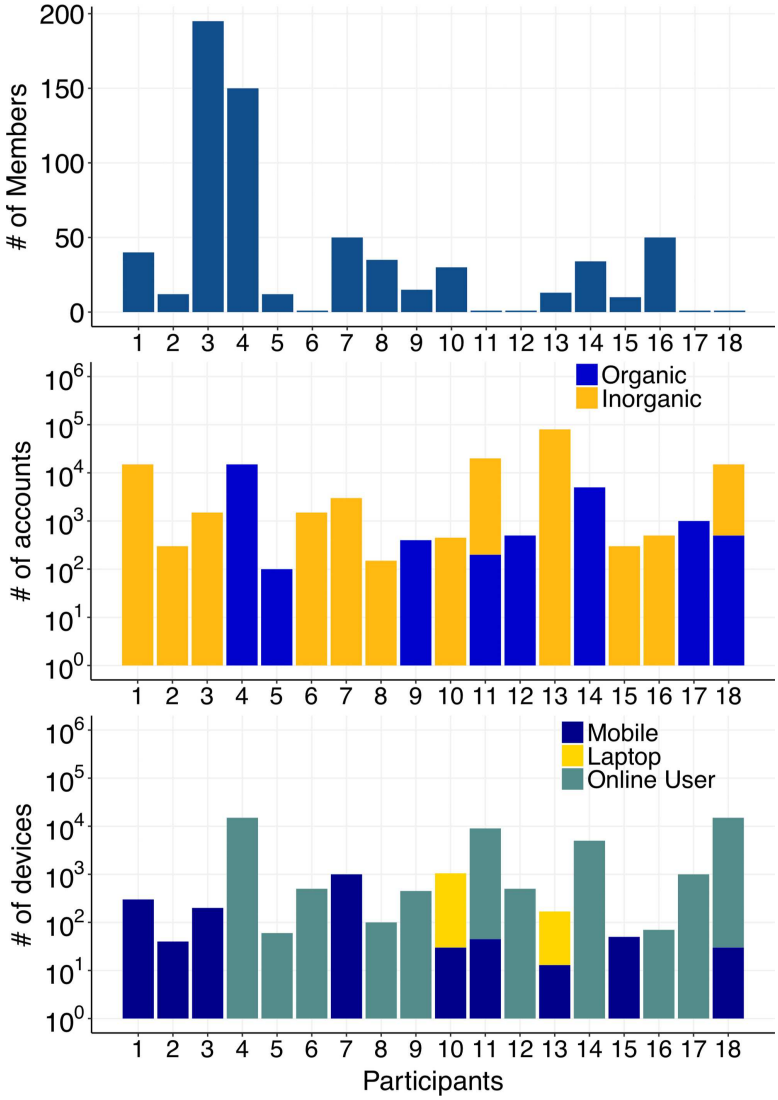


Figure 7.3: Participants profiles. Number of team members, number of accounts, number of devices. (top) Number of team members of organizations claimed by the 18 participants. 13 participants mentioned a team with more than 10 members. (middle) Number of accounts claimed to be controlled by participating fraudsters, including organic and inorganic. (bottom) Number of devices claimed to be controlled by fraudsters.

past month (M = 34.11, SD = 18.37). They claimed to charge between $0.5 and up to $6 per posted review (M = 2.16, SD = 1.86).

7 participants, who claimed to control thousands of accounts, also claimed to be able to write an "unlimited" number of reviews for a single app, i.e., more reviews than the developer can ask or afford. The other 12 participants with up to 3,000 accounts claimed to be able to write a number of reviews that was consistent (smaller or equal) to the number of accounts that they previously claimed to control.

Finding 3: PA.1 (Fraud types) is correct in that the expert fraudsters have many accounts and participate in many jobs.

7.5.3 Fraudster Capabilities: Devices

All the participants claimed to own or have access to multiple, real, mobile devices. Figure 7.3(bottom) shows the number of devices claimed to be controlled by each of the 18 participants. 3 participants (P5, P10 and P13) claimed to also use virtual devices (i.e., mobile device emulators running on laptops). For instance, P13 claims to have 13 laptops and use the Bluestacks emulator to install and review apps, and also 13 smartphones.

Device types. Several participants (P4, P5, P6, P8, P9, P11, P12, P14, P17, P18) have access to organic communities of users, thus to a diverse set of devices. 4 participants (P1, P10, P11, P13) claimed to own only low-end, cheap devices. Others (P7, P15, P16) claimed to own a mix of low, medium and high-end devices, dominated by low-end devices. For instance, P7 claimed that most of his 1,000+ devices are low-end Xiaomi, Micromax, Lenovo. However, a few devices are high-end (e.g., Samsung), which are needed in order to review virtual/augmented reality apps.

Device source. Most participants claimed to purchase their devices on the regular market. However, P11 claimed to have purchased his 45 devices, cheaply, on the black

market, and P18 claimed to run a mobile device repair shop, and use the devices he is supposed to repair (around 30 at any time) to write reviews.

Device storage. 6 participants claimed to store the devices on a table, easily accessible. P1 claimed to store the devices in a separate room. P7, claimed that their more than 1,000 devices are stored on a different floor, where the "install and review" team is located. However, they keep their high-end devices safe in a locker after use.

We also asked P7 about how they manage to charge 1,000 devices. P7 claimed that they have a dedicated team to manage all the devices, and charge a device every 2-3 days. Further, he claimed that they keep the devices on during office time, and switch them off after 11pm-midnight.

App-device compatibility issues. When asked what do they do when they need to promote an app that is not compatible with their devices, 9 participants (P5, P6, P8, P10, P11, P13, P14, P16, P17) said that it never happened. However, P7 said that he runs campaigns to recruit fraudsters who own compatible devices, or even purchase such devices. P9 and P15 said that they provide as many reviews as they can from their compatible devices, and contact the developer to explain the problem. P12 skips the job.

Finding 4: Not all participants claim to have only cheap low-end devices. Some participants claimed to have also medium and high-end devices. Others report ways around device cost limitations, e.g., to outsource jobs when they need device types that they do not have.

7.5.4 PA.4: Lockstep Behaviors

9 participants claimed to work on multiple apps at the same time, at least 4-5 (P6) and up to 10-15 (P13).

Several participants claimed to maintain a spreadsheet of accounts and devices used for each job, and use it to select the accounts and devices for the current job. 5 participants (P5, P7, P10, P13, P18) claimed to select the devices for a job, in a sequential, round-robin manner. 6 participants (P5, P7, P13, P15, P16, P18) claimed to select the accounts from which they write the reviews, in a sequential manner. For instance, P7 chooses both accounts and devices that were least recently used, by using the same account to write a second review only after using all his other 3,000 accounts. P15 said that they choose the least used accounts, to balance account usage and ensure that no accounts are left unused.

P16 said that they monitor their accounts in terms of number of their reviews filtered by Google, and choose accounts based on their filtering performance.

3 participants (P6, P8, P9) said that since they send the jobs down the hierarchy, the choice of device and account is made by their remote online employees.

Accounts Per Device. Several participants said that they login into multiple user accounts from the same device. Some participants claim that they do not post more than one review from the same device to an app, while others do not report this constraint.

P5 uses a fixed set of 2-3 accounts to login on one device at a time, then uses those accounts to review multiple apps. However, he also claims that he does not provide more than one review from one account for an app. P6 said that he instructs his remote workers to login on at most 2 accounts from any device (at a time), however, they can review the target app from both accounts. P7 mostly use 2-3 accounts from a device for safety. P8 claims to login to 5 accounts on his device, and his Whatsapp group members login to 3-5 accounts per device. P9 claims that he has logged into 4 accounts in a device, but he does not allow his workers to post more than one review from any device. P15 and P16 and their teams, keep track of which accounts they use to login on any device. Once they log out from an account, they log back in 7-10

days. P10, P11, P13, P18 said that they only login into one account at a time, on any device that they control. P11 motivated this choice

"If we provide multiple reviews from one device, Google will keep only one review for that device (sic)".

P10 said that before logging in to an account on a device, his physically co-located team flushes the virtual device and changes its MAC address. After using the account and virtual device pair for a few days to install and review several apps, the team logs out and repeats the process. They then leave this account unused for 1-1.5 months. The reason cited for this behavior is that after that interval, Google does not check that the new login is from the same MAC address as the previous one. P13 and his team also stay logged in on the user account for 3 days, then they reset the device (also using cccleaner) before logging in to the next account. P18 has one device for each account that he controls, and a 1-to-1 mapping between accounts and devices.

Strategies. Participants also reported unexpected strategies:

• *Layered jobs*: In addition to his own workers, P7 also organizes fully automated CPI (Cost-Per-Install) campaigns to provide incentive to other workers to install the target app, and CPA (Cost-Per-Action) campaigns to provide additional incentive for using the app. P12 employs a different strategy depending on the number of reviews required by the job. For up to 50 reviews, he uses specialized review-exchanging Facebook groups (§ 7.2). For 50-100 reviews, he also outsources the job to individual Facebook group workers. For more than 100 reviews, he also hires microworkers [mic, rap]. Similarly, P18 can provide up to 30 reviews. If more are needed, he shares the target app link with his Facebook group.

• *The device repair shop ploy.* P18 claims to run a mobile device repair shop and to use the mobile devices that he is supposed to repair, to post reviews.

• *Outsourced review slots*: P14 controls the time when each online team member is supposed to write the review for a target app. P17 posts the link to the target app,

to the 20 Whatsapp groups he controls, sequentially, and stops posting when enough reviews have been posted. P18 also claims to impose a 2 hour gap between reviews provided by remote Facebook group workers.

Finding. We found that some workers are still hard-core lockstep even reusing the accounts in the same order for the apps they review, whereas other workers show a more random behavior perhaps due to the organic nature of the fraud, i.e., fraud arriving from hybrid accounts: real users that eventually engage in opinion spamming.

7.5.5 PA.5 & C.2. The Art of Evasion

Awareness of Fraud Detection. Most participants said that it is very infrequent for their accounts to be deleted. P2 reported that *"Sometime the email might be desabled, in that case the review will be shown as "a Google user"."*

All participants reported that Google has deleted some of their reviews. Most report deletion as a small or negligible percentage (under 5%) of all reviews posted. However, P5, P9, P10, P14 reported 10-20% of their reviews being deleted. P3 stated that *"I have to ensure the buyer that after 24 hours they will have the required amount after deletion. For that I have to check and provide some review again if some are deleted."* P7 provide guarantees of reviews sticking for 5-7 days and refill deleted ones for free. Surprisingly, only P6 reported the review deletion percentage to depend on the app, ranging from 2% on some, to 30% on others.

Perceived Reasons for Deletion. Participants reported diverse reasons for deletion:

• *Device re-use*. P5 and P10 blame it on using the same device to write multiple reviews for an app: *"One of the main reason to erase the reviews is using the same device. I always track the screenshot that my workers provide me for their work proof. If I see two or more reviews from one worker have been deleted, I am pretty sure that*

they have used the same device for those reviews, I'll warn him for that, and next time I'll not assign him any job (sic).

• *Improper VPN use.* P10 blamed it on the use of VPN: *" VPN is one of the main problems. One safe way is, login from normal IP, then write review from VPN. If you login using VPN, Google will detect this as fraud."*

• *Improper app use.* P12 said that Google deletes reviews if the users *"do not care to use the app and keep it installed for more days."* More details in § 7.5.6.

• *Extended account use.* P3, P9, P18 report that using the same account to write many reviews in a short time, may trigger Google's alarms. To avoid this, they claim to stop using an account for several months (1-3) after using it to post 10-12 reviews. This is consistent with earlier participant claims of limited account use, see § 7.5.4.

• *Misfires of Google fraud detection.* P6 blames it on Google: *"Sometimes genuine reviews get deleted and sometimes multiple reviews from same devices don't get deleted. It varies from app to app, as well as the mood of Google."*.

PA.5.1. Noisy Reviews. 8 participants (P2, P3, P5, P7, P10, P13, P15, P16) said that they do not review other apps to avoid detection. Of the physically co-located teams, only P1 said that they review products for which they have not been hired, which they pick at random. 7 participants with online and organic team members (P4, P6, P8, P11, P14, P17, P18) admitted that their online team members review other apps for which they have not been hired, which they normally use in their real life. P4 said *"That's why we are using real/organic users account. We don't need to follow any strategy. The real users' behaviors serve the purpose of authenticity. We always instruct them to use other popular apps from their accounts."*

PA.5.2. VPN Use. 10 participants (P4, P6, P7, P8, P10, P11, P12, P14, P17, P18) claimed not to use VPNs. 5 participants (P1, P3, P5, P13, P15) admitted to using VPNs. P3 claimed to use VPNs or proxies occasionally, based on job specifications:

"We use VPN or proxy only when it is required in the job specification. For example, if I need to install from USA, we have to use USA proxy server.(sic)"

PA.5.3. Emulators: 2 of the 18 participants (P10 and P13) said that their teams use virtual devices running in laptops. The others use mobile devices or have access to real users equipped with mobile devices.

PA.5.4. Upvoting: Fraud and accomplice. 6 of the 18 participants (P5, P7, P10, P14, P15, P16) said that they upvote reviews written by their team from other accounts. P7 said *"We upvote the reviews put by our team and also other reviews which are positive."*

Downvote Negative Reviews. P10 said that his team downvote negative reviews of the apps they target, in order to trigger Google's filtering mechanism, thus remove those reviews. P7 said *"We provide upvote and downvote services to make positive reviews on top and negative in bottom."*

C.6. Account Blending: 7 participants (P1, P2, P7, P10, P11, P13, P15) said that they worked on jobs where they only used fresh accounts. P10 said that *"We do it because Google always keeps the reviews received from new accounts."*. 5 participants (P1, P2, P7, P16, P18) said that they use a mix of old and new accounts. P1 claimed to use a 50%-50% split of old vs. new accounts in a job, if the developer does not make a specific request. P2 claimed that it depends on the job requirements, but they use a 60%-40% split. P7 claimed to use mostly old accounts. P16 claimed to use a random mix of new and old accounts, on all their jobs. P18 said that *"When I create new accounts I always use them with old accounts."* 12 participants had seen jobs that required only the use of old accounts, with P4 claiming 6-8 such jobs in the past month.

C.2.2. User Account Validation. P2 and P3 said that they prefer to use e-mail addresses to validate user accounts. However, P3 mentioned that Google may force them to use phone numbers for validation. All other participants use phone numbers

to validate accounts. Only P16 claimed that *"we use virtual numbers and Google accepts them.* All other participants said that they need to use real phone numbers.

Real phone numbers require access to SIM cards, which can be expensive. However, participants revealed ingenious solutions to bypass this limitation to be able to validate hundreds of users accounts using only a few working phone numbers. P1 said that *"it is possible to verify many accounts with 1 phone number."* P3, P10, P11 and P17 claim to use friends and family: P3 said that *"We use our friends and family phone numbers. For example, I meet a friend on the road, I ask him to check the message and I use his phone number to verify an account."* P10 further said that *"In Bangladesh one person can buy as many as 20 SIM cards using his credentials. [..] For example, for my 450 Gmail accounts I have used at least 200 phone numbers."* P5 mentioned that he borrowed SIM cards from friends. P7 and P15 refer to a phone number verification service: P7 said that *"we pay other people to get a one-time code from their mobile SMS to verify those accounts"*, while P15 said *"we use a person who has lots of phone numbers and provides a service to verify Gmail accounts with phone number."*

P13 said that they purchase user accounts that are already validated.

Some participants reported limitations on the reuse of phone numbers. For instance, P3 and P8 said that one number could be used for 3 - 5 accounts but not immediately, while P1 said that *between two verification using the same number, we have to wait at least 3 months."*

7.5.6 C.7. App Installation and Use

C.7.1 and C.7.2. Delayed Review, App Open and Usage. 14 participants claimed to wait, open or use the app before reviewing it. P5 and P9 wait a few hours before reviewing the installed app. P9 also claimed to use it for 5-10 minutes. P6 and

P8 claimed to open the app 1-2 times before reviewing. P7 claimed to use the app as a normal user. P10, P13 and P16 claimed to keep the app open for 7-10 minutes, 3-4 minutes and 10-15 minutes respectively, before writing the review. P12, P14, P17 and P18 claimed to recommend to their online and organic teams to open the app for a few minutes and even use it before reviewing. P4 said *"We try to navigate all the pages of the app before writing the reviews."*

C.7.3. App Retention: All participants admited to perform retention installs. In fact, P10 said that this is required to prevent review filtering: *"Google takes 72 hours to verify the review. If you delete this app in this period, Google will drop the review."*

P1, P5 and P15 said that they keep the app for 1 day after reviewing it. P5 said however that he charges extra when developers ask them to keep the app for 2-3 days. P2 claimed to keep the app between 7 days and up to 2 months. However, most participants said that they keep the app for a few days, ranging from 2-3 days (P4, P8, P10, P13, P14) to 1-2 weeks (P17). Consistent with previous claims, P10 said that *"We keep the app 3 days in the virtual devices. After that we flash the virtual devices and change the MAC address and use it for the next reviews."* P4 said that his workers keep the app installed until they need the space.

Review Without Install. When asked, 4 participants (P5, P10, P13 and P18) said that one can review an app without prior installation of the app from a device on which the account is logged in: *"Click on install then stop installing immediately. The app would not be installed but it will allow us to write reviews."*

Findings. While several participants reported the ability to bypass Google's sanity check of preventing reviews without prior app instalation, all the participants suggest not only installing but also keeping the app installed for days after the review.

7.5.7 PA.2. Fraud Time Points

PA.2.1. Early Bird Fraud. 14 participants said that they have seen and have worked on apps that were recently launched. They said that either the hiring developer mentions that the app was recently launched, or that they infer this information based on the app status when posting their first review. Only P1 and P11 said that they have seen only 1-2 such jobs in the past month, while 3 participants (P9, P10, P13) claimed to have worked on 20 or more (up to 40) such jobs in the past month. P7 claimed that "We even work on apps which are going to be launched soon. A few of our clients rely on our agency from pre-launch to launch and then post-launch (sic)".

PA.2.2. Proactive Fraud: 15 of the 18 survey participants said that they were hired to proactively inject fraud to maintain a desired average rating of an app. However, participants said that they have seen very few such jobs, and worked on even fewer. 6 participants had worked on 1 such job in the past month, 3 participants worked on 2, and 3 worked on 3-4 such jobs last month. One participant had worked on only 10-12 such jobs in total. This is even the case for participants who offer such packages. For instance, P7 said *"Yes, we provide ASO maintenance services and subscription-based services for installs and reviews in which you can select packages, and you receive a fixed number of reviews and installs each month"*, yet they only worked on 3 such jobs in the past month.

7.5.8 PA.3. Review Burst Assumption

16 participants claimed to have seen jobs that specify how many reviews per day the workers should post. 2 participants claimed to have seen 40-45 such jobs in the past month, the others ranged between 1-30. P5 stated that most developers want a slow posting rate, e.g., 2-3 reviews per day, and to maintain this rate, the developers

provide the text of the reviews to be posted each day. P6 said that *"some developers with money don't care whether reviews stay or not. They just need the number of reviews in their apps, quality doesn't matter to them. They just want short-time business."* P10 said that *"Experienced buyers always know how many reviews are good each day. They mention like 2-3 reviews each day. We also agree on that."*

P1, P3 and P5 reported that they suggest to the hiring developers, the rate of posting reviews. P5 said *"If the developer asks for 30 reviews each day, I have to warn him that it's harmful to his app as Google may detect this as fake. Then I'll suggest to him that I will take 10 days to provide 30 reviews."*. Most participants suggest 2-3 reviews per day, but some (e.g., P11, P14, P17, P18) recommend higher numbers, up to 30-40 reviews per day (P14).

Review Rate Crescendo. Several participants suggested that the number of recommended daily reviews is a function of the app's existing review count. P6 said that *"for new apps with less installs, it is better not to provide many reviews each day. But for popular apps, 20-50 reviews each day would be acceptable."* P7 said that *"if your app has very fewer reviews, you should go with 3-5 reviews daily so that Google should not become suspicious. If someone has a large number of reviews, they can get 20-25 reviews daily."*

P10 revealed a different strategy: *"We provide a slow rate at the beginning. Like per day 1 review. Like 5 reviews in 6-7 days. After 10 reviews we start posting 2 reviews each day. After 150 reviews we can provide 3-4 reviews each day."*

7.5.9 C.3. Review Campaign Duration

All the participants except P4 said that they have seen ASO job posts that specify the required duration of the review posting campaign. P6 and P15 said that it is rare for developers to specify the duration of the campaign or the number of daily reviews,

and that developers are more concerned about the total number of reviews. However, P2 said almost all the jobs he has seen in the past month, mention the campaign length. In in the past month, 5 participants have seen 3-5 such jobs, 4 have seen 6-10 jobs, and 5 have seen 11-35 such jobs. The longest required interval reported by the participants for a campaign is 18 months. 12 participants reported longest seen required campaigns of 1-6 months, and 6 participants reported campaigns of 7-18 months.

Finding: Our survey suggests a strategy shift, driven both by a subset of developers and workers, who are aware that substantial review bursts may be more easily flagged. Specifically, some developers and fraud workers prefer to spread out reviews over longer campaigns. Such a strategy prevents the peer-review site from flagging review bursts and also ensures a consistent positive review campaign over a longer interval.

7.5.10 PA.6. Review Writing

Review Text Source. 2 participants (P3, P4) said that they always write their own reviews. The other participants said that they both receive or request the review text from the developer, and they also write their own reviews. P2 said that they receive instructions about the reviews from the developer. P11 reported developers who provide review samples, from which they are supposed to generate variations. 3 participants (P7, P8, P15) said that they either prefer or even ask the developer to provide the review text. P3 and P13 said that they study the app before writing the review. P13 claimed to ask the developer to provide the app's main features, which he uses to fabricate reviews.

PA.6.1. Review plagiarism: 8 participants (P1, P3, P5, P12, P13, P14, P15, P18) denied using plagiarism or self-plagiarism. Several participants (P2, P4, P6, P9, P11, P17) admitted to some form of plagiarism. For instance, P2 only does it when the

developer mentions the source, e.g., to copy reviews from other peer-opinion sites. P4 admitted that their reviews are sometimes similar, as they are short. P16 claimed to have a set of ready-made reviews which his team changes slightly for different apps. P9 said that *"Not exact copy-paste. But sometime we copy and modify reviews from other apps that are similar to this app. (sic)"*

PA.6.2. Review Length: One participant claimed to write reviews that have at least 10 words. Two participants claimed to write reviews that are 15-30 words long. Several participants admited that their reviews are short: *"We don't use many words or big sentences because Google may match the pattern. We always use short message like "Good app", "Awesome", "Fantastic". These are very common but easy to write and Google may not complain".*

Review Posting Process. The participants had a mixed strategy of typing the reviews directly on the device, vs. cut-and-pasting them from a separate source. 11 participants said that they type the reviews directly from their devices. P5 said that they cut-and-paste reviews if provided by the hiring developer, otherwise they type their own (short reviews). Several participants organize teams of remote fraudsters, thus cannot verify their actions. However, P7 noted that most devices do not allow cut-and-paste.

7.5.11 PA.7. Ratings

PA.7.1. Extreme Ratings. All participants admitted writing mostly 4 or 5-star reviews unless they receive special instructions from the developers. 8 participants (P3, P7, P8, P9, P10, P11, P14, P18) said that they receive instructions on the ratio of review ratings from the developers. P12 said that these instructions can also include ratings lower than 4 stars: *"Developers request us to write a few 4, 3 and even few 1 star reviews."*

When there are no instructions on the rating distribution, several participants claimed to maintain their own ratio. For instance, P5, P16, P17 claimed to post a 10%-90% ratio of 4 to 5 star reviews, P2, P10, P18 have a 20%-80% ration, P1, P9 have a 30%-70% ratio and P13 has a 40%-60% ratio. 3 participants (P6, P11, P14) said that they do not maintain any specific ratio, while P4 and P12 post only 5-star reviews.

3 participants claimed strategies to also post lower ratings. For instance, P6 mentioned that: *"If the average rating goes up to 4.3 or 4.4, I also write a few 3-star reviews along with 4 and 5-star reviews."* P7 said that *"When posting more than 200 reviews, we suggest to the client to have at least 5 to 6 reviews with 3 star ratings."* P15 claimed to post a 10%-30%-70% ratio of 3, 4, 5-star reviews.

Findings. While fraudsters claim to and post mostly 4 or 5-star reviews, several also report evasion strategies, where they post also a few neutral and negative ratings.

7.5.12 PA.8. Review Bots

5 participants (P1, P11, P13, P15, P16) said they access their accounts regularly. 12 participants said that they access them manually. However, 3 participants (P13, P15, P16) said they use scripts and automatic login systems to periodically access their accounts, keep them alive, and report if any are inaccessible. All the participants who organize organic users said that organic users access their accounts regularly for their regular use.

All of the participants said that they write their reviews manually, and do not use any script for this purpose.

Several participants said that they use specialized programs to help with installs, and manage jobs accros multiple team members. For instance, P6 said that they use install bots if the client agrees. P7 said that they have special apps (names not

provided due to privacy concerns) that they use to push jobs to their workers, and manage the jobs and pay for the work. P9 uses an "admin panel" app to contact his team admins who respond with the number of reviews that their sub-teams can provide.

Finding. While 6 of the 18 participants claimed to use tools to keep alive their accounts or install apps, none used tools to automatically post reviews.

7.5.13 Account Life Cycle

New Account Creation vs. Purchase. 6 participants (P1, P3, P7, P10, P13, P16) claimed to create new accounts periodically, ranging from once a day (P10) to once a month (P16). P2 and P9 claimed to create new accounts when they don't have enough accounts for a job, especially when the job requests accounts from a specific geographic region. P5 and P18 create new accounts when Google deletes some of their accounts. P15 create new accounts when the job requests more reviews than they can provide. Only 5 participants (P1, P5, P13, P17) admitted to purchase new accounts. P1 claimed to have purchased more than 10,000 accounts, while P13 claimed to have purchased 47,000 accounts. Two participants (P1 and P3) volunteered the fact that they age their new accounts (1-2 months) before using them to post reviews.

PA.9. Account Name Variation. 13 participants mentioned they use fake name generators, e.g., [FNG], to get random names for their accounts. Some of them create account names to correspond to specific geographic regions, as sometimes also requested by developers. P2 even claimed to send the chosen names of to the employer for feedback. P11 claimed to use random names from Google search and P7 said that they have their own name database. P4, P7 and P14 said that their use of organic fraudsters ensure that they use real user names.

7 participants mentioned that they add profile pictures, which they retrieve from different sources, e.g., Google search, Google Plus, pixabay.com, to make the account look more authentic. P9 said *"After we use fake name generator to create the account name, we search the name in Google Plus and choose a profile, then we choose a random person from the list of followers and use his image for the account profile."* P10 however said that they do not use any profile picture as pictures define the demographics of the profile, and developers do not want to reveal this information.

PA.10. Singleton Accounts. 6 participants (P1, P2, P7, P10, P13, P15 admited to having worked on jobs that required them to create accounts just to post one review and then to abandon them. P1 and P2 said that the cost for posting such a review is higher than usual, $8 and $10 respectively. The reason for this is due to the effort to create an account, which will not be amortized over multiple fake review posting activities. The reason given by the participants for being requested to do this is that Google does not filter reviews posted by singleton accounts, since its fraud detection module needs more information to build a reputation for the account.

Account Abandonment. With the exception of the above singleton account jobs, 9 participants said they have never abandoned an account. 6 participants claimed that they abandon their account only if Google blocked that account for suspicious activities. P17 and P18 said that they abandon an account only if its reviews are continuously deleted by Google. P2 said that *"I remember I used some accounts for many different jobs, like YouTube review, Google Place review, app review and other services. Google disabled those account for misuse. After that, we dedicated accounts for app reviews."* Consistent with statements about their account use strategies (§ 7.5.4), P3 and P9 said that they never abandon accounts, but only stop using them for a few months after using them a few times.

7.5.14 C.9: Proof of Work

12 participants said that they use screenshots as a proof of work. 5 participants said that they send the usernames of accounts that they used to post reviews. P6 claimed to send permalinks to their reviews: *"Normally I check my reviews for 2-3 days and then send the permalinks that are direct links of the review I post, or names I used to post the reviews."*

Work verifications can take place at the team level. For instance, P3 said that *"[..] we ask everyone to post reviews in the team. Then I track how many reviews we provide and they also send me the screenshot. If the buyer requires the screenshots I send him those too."* He also said that *"Sometimes, the developer keeps track of the reviews we post, and gives us 24 hours to show that the reviews are alive. If any review is deleted during this time, we have to re-post the reviews."*

P6 said that *"If we get a report that any review is being deleted then we check that user's mobile and ask him to provide screenshot of the app installed immediately. If he fails to provide that, I flag him as a bad user and we consider him less for the next tasks."*

P9 verify that their team members do not post multiple reviews from the same device: *" I always check the screenshots and see if they are taken from the same device. One clue is, if the status bar of the phones of two screenshots from the same admin are similar, I consider those reviews to be from the same device. We believe that each device status bar is different as different people run different services in the background."*

Findings. Some developers require and verify that fraudster reviews are not filtered for days after they are posted. The distributed nature of developers and fraudster teams imposes interactive work verifications. Teams also verify the work claims of their members, and punish cheaters.

7.6 Recommendations

We discuss several defenses that the findings in this paper could enable, to help peer-opinion sites identify and even prevent fraud, or at least increase its cost.

Keep Account Owners Busy. Peer-opinion sites could impose small account management tasks at random times (e.g., change password every 3 months). This will require fraudsters who own many accounts to update their password files ...

Account Verification. Several fraudsters mentioned using the same SIM card to validate multiple accounts, albeit after waiting for a few months. Peer-opinion sites could ask users to re-validate their accounts at later, random login times, especially if the SIM cards used to validate them have been used also for other accounts.

Organic vs. Inorganic Reviews. To distinguish between fraud workers who provide inorganic from those who provide organic reviews, we can use the diversity of the device types used to post those reviews. This is because organic workers organize real users who user their personal user accounts and devices to post reviews, thus have more diversity than inorganic workers who control fewer devices than accounts, often of the same brand.

App Use. Most fraudsters suggest that it is better to use the apps before reviewing them, to mimic genuine behaviors. They claim to open the app, interact with it for a few minutes or hours and then post the review. We believe that the time between installing the app and reviewing it must be substantially different when reviews are posted from honest and fraudulent accounts. This is one feature that Google's anti-abuse techniques can adapt to help detect fraudulent reviews and accounts. Similarly, features such as the number of times the app has been opened before the review and the length of each session can aid in this purpose.

Review Process. Most fraudsters mentioned that they keep the app installed for at most two weeks after they reviewed it. This behavior can also be exploited by

Google's techniques to detect abuse as we argue that the natural flow of the review process should be more linear: a user first install the app, then review it, after he has used it for some time. It is suspicious if an app receives a review soon after was downloaded and even more if it is uninstalled right after a positive feedback. We could design a system that keeps the review posted and detect whether the user uninstall the app soon after it was reviewed positively. If so, this review is likely to be fake.

Delayed Removal of Fraud. Some fraudsters claimed that reviews written from the same device are likely to be deleted by Google and that they avoid doing it. In addition, some fraudsters mentioned that reviews via VPN are blocked by Google and that account activity is important to blend in as honest users. In other words, some immediate countermeasures may serve as an oracle of what actions Google can detect as abusive. The fraud workers can then use this oracle to test and adapt their strategies to learn and bypass these countermeasures in a short period of time. We propose to postpone the removal of fraudulent content so that it is hard for fraudsters to reverse engineer Google's engine. This approach proved to be hard to notice by fraudsters in Instagram using only one-day delay removal.

Mislead Fraudulent Accounts. Similar to the previous recommendation, we suggest to mislead fraudulent accounts by showing both fake and honest actions to fraudulent users, but only genuine content to honest accounts. In this sense, we propose to select different content to fraudulent accounts in order to mislead them into believing that their actions are being effective. This will make it harder for them to reverse engineer Google's efforts.

7.7 Conclusion

We survey the assumptions made by the fraud detection research community on the capabilities and behaviors of fraudsters and study their relevance to app search optimization fraud conducted for Google Play apps. For this, we designed and conducted in-depth interviews with expert fraudsters recruited from several freelancing sites. In addition, we have validated these interviews results with direct empirical data collected from the social networking site, establishing ground truth for future research. Furthermore, we have identified novel assumptions that seem critical to the detection and de-anonymization of the fraudulent communities studied. We have similarly analyzed and validated these new assumptions which provide novel insight into the techniques and methods used by fraudsters in the wild.

7.8 Survey Questions

Screening Questions.

1. How many user accounts do you control in Google Play?
2. For how long have you been active doing App Store Optimization for Google Play?
3. How many jobs have you worked on Google Play review approximately?
4. How much do you charge for a review? How much do you charge for an app install?
5. Can you tell me what types of ASO services do you provide in Google Play? *If the participant is confused, suggest Ratings, Reviews, and App Installs.*

Work's Requirement.

1. Have you seen jobs that tell you for how long to post reviews, that is, for how many days?

   If the answer is yes:

   (a) How many such jobs have you seen in the past month?

   (b) What is the longest such time interval that you have seen in a job?

2. What do you think is the largest number of reviews that a freelancer could write for a single app?

3. Have you seen jobs that ask you to post a certain number of reviews per day?

   If the answer is yes:

   (a) How many such jobs have you seen in the past month?

   (b) What is the average number of reviews that you think is better to post in a day for a single app?

   (c) Do you post them all at the same time, or at different times during the day?

4. Do you know any developers/employers who ask freelancers to post reviews for recently launched apps?

   If the answer is yes:

   (a) How many such apps have you seen in freelancing sites in the past month?

5. Were you ever re-hired by the same developer to review the same app at a later time? For instance, 2 weeks or 6 months after the first job?

   If the answer is yes:

   (a) How many times?

   (b) Were you ever hired by a developer to review more than one app?

6. Do you provide services that ensure that the rating of the app will stay above a threshold, for instance, above 3.5 or 4 stars? How much do you charge for this service?
7. Have you done any job for which you have not received any payment?

Device Usage.

1. What devices do you use frequently?

   If they mention laptop:

   (a) Do you ever/do you prefer to use a browser to write reviews? or do you use emulator? What about virtual machines?

   (b) What about scripts?

   If they mention mobile:

   (a) How many mobile devices do you have?

User Accounts.

1. Some people think it is important to create new user accounts when they start a new job. How many new accounts would you create on average per job?

   If they don't give a good answer then ask:

   (a) Can you give me one recent example?

2. Is there any effective way to pick a good name for a user account?

   (a) Are the names that you choose very different from each other?

   (b) Do you ever use a random name generator?

3. One participant mentioned that they have purchased user accounts from a third party. Have you ever done that?

   If the answer is yes:

(a) Do you purchase such accounts when you start a new job?

(b) Do you purchase such accounts at random times?

(c) How many new accounts do you think you have purchased in total?

(d) Have you noticed if any of the accounts that you purchased, had been used already, for instance, to review other apps?

(e) Do you purchase authentic user accounts which were used for good purpose before?

4. Do you ever need to verify newly created Google Play accounts?

   (a) What is the preferred way to verify those accounts? Email or Phone?

   If phone:

   (a) Do you buy and use virtual phone numbers, like Twilio or Google Voice?

5. Do you access your Google Play accounts regularly? How do you do that?

   (a) Do you access them manually?

   (b) Do you use any script to access and maintain their accounts?

   (c) Is it necessary to use proxy or VPN services to access accounts?

6. Since freelancers have so many user accounts in Google Play, what are common strategies to remember passwords for these accounts?

   (a) Do you write the passwords down?

   (b) Do you use the same password for multiple accounts?

   (c) Do you use passwords that are easy to remember?

7. For how long do you usually use a Google Play account on average? Do you ever abandon a user account that you use in Google Play?

8. Do you ever create an account just to post one review and then you abandon the account?

Camouflage Strategies.

1. Did you ever have a Google Play account that was deleted by Google?
   (a) How many times did this happen to you in the past two weeks?
2. Has Google Play ever erased reviews that you posted?
   (a) Do you remember how many times this happened in the past 2 weeks?
   (b) From your experience, how long does it take for other people to see your reviews?
3. Do you know any strategy that people use to convince Google Play not to block their accounts?
   If the answer is yes:
   (a) Can you describe any strategies that people follow?
   (b) Do you write reviews for apps that you were not hired to review?
   If the answer is yes:
   (c) How do you select those apps?
   If not making progress (make a note of it), and ask:
   (d) Do you write reviews for apps that you personally like?
   (e) Do you write reviews for apps that you pick at random?
4. Do you ever use only new accounts to post reviews for an app in a job?
5. Do you use a mix of old and new accounts to post reviews for an app in a job?
   (a) How often did you do this in your past 5 jobs?

6. Have you seen jobs that explicitly request to use only old accounts that have a good reputation?

   If the answer is yes:

   (a) How many such jobs did you noticed in the past month?

   If none:

   (b) What about the past 3 months, etc?

7. How much do you charge for a review written from a new vs old account?, is there any difference?

8. Do you write the text of the reviews yourself?

9. Do you ever get the review text from the developer?

   If the answer is yes:

   (a) How many times in the past 2-3 weeks?

10. Have you ever seen jobs that ask you how long the review text should be? Do you remember that value?

11. Have you seen people copy parts of reviews that they posted for other apps?

12. Have you seen people copy parts of reviews written by others?

13. Give us some example of review text that you write usually?

14. How long is an average review that you write, in terms of the number of words or characters?

15. Did you ever post 3 star reviews? How many of your reviews are 3 star reviews? Half of them, a quarter, ten percent, less than 10 percent?

16. Were you ever hired to write bad reviews, with 1 or 2 star ratings? Do you remember how many such jobs you did?

17. What percentage of your reviews had 4 or 5 stars?
18. Do you ever go back and change a review that you wrote in a job?

    If the answer is yes:

    (a) Can you tell me why?

    (b) Did you ever increase the start rating?

    (c) Did you ever decrease the rating?

    (d) Did you leave the rating the same but just change the text of the review?
19. Do you ask someone else to post the reviews that you wrote?
20. Do you ever use a script to post the reviews?
21. Do you know how to post a review for an app without installing the app?
22. How long do you keep the app installed after posting the review?

Collaboration.

1. What do you do when the number of reviews requested is higher than the number of accounts you have? Do you outsource the rest ?
2. Do you work individually or are you a part of a team or business?

   If the answer is yes:

   (a) If part of a team, how large is your team or business?

   (b) How do people in the team communicate?

   (c) How do they distribute the profit?
3. Do you maintain any work hierarchical levels in your team? For example, catching the clients, writing reviews, creating accounts?

4. Have you seen jobs where developers ask freelancers to collaborate with other freelancers?

   If the answer is yes:

   (a) How do you communicate with them?

5. Do you know freelancers who communicate with each other to post reviews for the same job? Do you know if freelancers communicate among them the times when they will post their reviews?

6. Do you know freelancers who hire others to help them with the jobs they get - like a subcontract?

7. Do you share Google Play accounts with other freelancers to help with App Store Optimization(ASO) jobs?

General Questions.

1. What's your age?

   (a) less than 18

   (b) 18 to 28

   (c) 28 to 38

   (d) 38 to 48

   (e) 48 to 58

   (f) 58 to 68

   (g) older than 68

2. What is the highest degree or level of education you have completed?

3. Do you have another job besides freelancing? Can you tell me what it is?

CHAPTER 8

Conclusion and Future Work

Online services that assemble and leverage public opinion on hosted products are central to numerous aspects of peoples online and physical activities. Every day, people rely on online information to make decisions on purchases, services, software and opinions. People often assume the popularity of featured products is generated by purchases, downloads and reviews of real patrons, who are sharing their honest opinions about what they have experienced. Reviews, opinions, and software are sometimes fake, produced and controlled by fraudsters. Some of them collude to artificially boost the reputation of mediocre services, products, and venues, some game the system to improve rankings in search results and some entice unsuspecting users to download malicious software.

The developers of top ranking products in peer-review sites like Google Play, Amazon, or Yelp receive higher rewards, that include direct payments and ad-based revenues. Statistics maintained by peer-review sites concerning user activities for a product (e.g., reviews, ratings, likes, followers, app install counts) are known to play an essential part in the product's ranking. This has created a black market for *search rank fraud*, mediated by an abundance of crowdsourcing sites. Specifically, crowdsourcing fraudsters create or purchase hundreds of user accounts in the peer-review site, then post activities for the products of developers who hire them, from the accounts they control. Discouraging search rank fraud is essential to ensure trust in peer-review sites and the products that they host. Previous work in this area has focused on fraud detection.

In chapter 3, we studied temporal patterns in Google Play, an influential app market. We use data we collected from more than 160,000 apps daily over a six month period, to examine market trends, application characteristics and developer behavior in real-world market settings. Our work provides insights into the impact of developer

levers (e.g., price, permissions requested, update frequency) on app popularity. We proposed future directions for integrating analytics insights into developer and user experiences. We introduced novel attack vectors on app markets and discussed future detection directions.

In chapter 4, we have introduced FairPlay, a system to detect both fraudulent and malware Google Play apps. Our experiments on a newly contributed longitudinal app dataset, have shown that a high percentage of malware is involved in search rank fraud; both are accurately identified by FairPlay. In addition, we showed FairPlay's ability to discover hundreds of apps that evade Google Play's detection technology, including a new type of coercive fraud attack.

In chapter 5, We have introduced the concept of real-time fraud preemption systems, named as the FraudSys, that seek to restrict the profitability and impact of fraud in online systems. We propose and develop stateless, verifiable computational puzzles, that impose minimal overheads, but enable their efficient verification. We have developed a graph based, real-time algorithm to assign fraud scores to user activities and mechanisms to convert scores to puzzle difficulty values. We used data collected from Google Play and Facebook to show that our solutions impose significant penalties on fraudsters, and make fraud less productive than Bitcoin mining.

In chapter 6, We introduced the fraud de-anonymization problem for search rank fraud in online services. We have collected fraud data from crowdsourcing sites and the Google Play store, and we have performed user studies with crowdsourcing fraudsters. We have proposed DOLOS, a fraud de-anonymization system. DOLOScorrectly attributed 95% of the fraud detected for 640 Google Play apps, and identified at least 90% of the workers who promoted each of 87% of these apps. DOLOS identified 1,056 out of 13,087 monitored Google Play apps, to have suspicious reviewer groups, and revealed a suite of observed fraud behaviors. DOLOS significantly outperforms adapted dense subgraph detection and loopy belief propagation competitors, in two

coverage scores that we have developed.

In Chapter 7 we survey the assumptions made by the fraud detection research community on the capabilities and behaviors of fraudsters and study their relevance to app search optimization fraud conducted for Google Play apps. For this, we designed and conducted in-depth interviews with expert fraudsters recruited from several freelancing sites. In addition, we have validated these interviews results with direct empirical data collected from the social networking site, establishing ground truth for future research. Furthermore, we have identified novel assumptions that seem critical to the detection and de-anonymization of the fraudulent communities studied. We have similarly analyzed and validated these new assumptions which provide novel insight into the techniques and methods used by fraudsters in the wild.

BIBLIOGRAPHY

[AC08] Ahmed Abbasi and Hsinchun Chen. Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace. *ACM Transactions on Information Systems (TOIS)*, 26(2):7, 2008.

[ACF13a] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion Fraud Detection in Online Reviews by Network Effects. In *Proceedings of AAAI ICWSM*, 2013.

[ACF13b] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. *Proceedings of ICWSM*, 2013.

[AM12] Michael Anderson and Jeremy Magruder. Learning from the crowd: Regression discontinuity estimates of the effects of an online review database. *Economic Journal*, 122(563):957–989, 2012.

[and] Getting started with Android Studio. http://goo.gl/wgeUok.

[And11] Android Market API. `https://code.google.com/p/android-market-api/`, 2011.

[Ank13] Google I/O 2013 - Getting Discovered on Google Play. `www.youtube.com/watch?v=5Od2SuL2igA`, 2013.

[App16] Apptamin. Optimize Your Google Play Store App Details Page. `http://www.apptamin.com/blog/optimize-play-store-app/`, 2016.

[AR16] App Reviews. `http://www.app-reviews.org`, Last accessed November 2016.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[AS16] App Such. `http://www.appsuch.com`, Last accessed November 2016.

[ATK15] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.

[Att] A.G. Schneiderman Announces Agreement With 19 Companies To Stop Writing Fake Online Reviews And Pay More Than $350,000 In Fines.

[AV16] Apps Viral. `http://www.appsviral.com/`, Last accessed November 2016.

[BA13] Sajid Yousuf Bhat and Muhammad Abulaish. Community-based features for identifying spammers in online social networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 100–107. IEEE, 2013.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of CRYPTO*, 1996.

[Ben88] Y. Benjamini. Opening the box of a boxplot. *The American Statistician*, 42(4):257–262, 1988.

[Ber96] Dan J Bernstein. Syn cookies, 1996.

[Bes] BestAppPromotion. `www.bestreviewapp.com/`.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly, 2009.

[Bor06] Nikita Borisov. Computational puzzles as sybil defenses. In *Proceedings of the IEEE P2P*, 2006.

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[BRO17] RICK BROIDA. Spot fake Amazon and Yelp reviews on the go with Fakespot for iOS. CNet, 2017.

[BS00] E. Brynjolfsson and M.D. Smith. Frictionless commerce? a comparison of internet and conventional retailers. *Management Science*, 2000.

[BSLL+16a] Prudhvi Ratna Badri Satya, Kyumin Lee, Dongwon Lee, Thanh Tran, and Jason (Jiasheng) Zhang. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2365–2370, 2016.

[BSLL+16b] Prudhvi Ratna Badri Satya, Kyumin Lee, Dongwon Lee, Thanh Tran, and Jason Jiasheng Zhang. Uncovering fake likers in online social networks. In *Proceedings of the ACM CIKM*, 2016.

[BSLL+16c] Prudhvi Ratna Badri Satya, Kyumin Lee, Dongwon Lee, Thanh Tran, and Jason Jiasheng Zhang. Uncovering fake likers in online social networks. In *Proceedings of the ACM CIKM*, 2016.

[BXG+13] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In *Proceedings of the WWW*, 2013.

[BZNT11] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: Behavior-Based Malware Detection System for Android. In *Proceedings of ACM SPSM*, pages 15–26. ACM, 2011.

[Cal16] John Callaham. Google is finding and taking down fake reviews in the Play Store. Android Authority, 2016.

[CCMT90] R.B. Cleveland, W.S. Cleveland, J.E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[Cha00] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Workshop on Approximation Algorithms for Combinatorial Optimization*, 2000.

[CHZY17] Hao Chen, Daojing He, Sencun Zhu, and Jingshun Yang. Toward detecting collusive ranking manipulation attackers in mobile app markets. In *Proceedings AsiaCCS*, 2017.

[Cip16] Jason Cipriani. Google starts filtering fraudulent app reviews from Play Store. ZDNet, `https://tinyurl.com/hklb5tk`, 2016.

[CNW+11] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the SIAM SDM*, 2011.

[CSN07] A. Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *Arxiv preprint arxiv:0706.1062*, 2007.

[CYYP14] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 477–488. ACM, 2014.

[DCFJ+14] Emiliano De Cristofaro, Arik Friedman, Guillaume Jourjon, Mohamed Ali Kaafar, and M. Zubair Shafiq. Paying for likes?: Understanding facebook like fraud using honeypots. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 129–136, 2014.

[DG08] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of CRYPTO*, 1992.

[Dol] Dolos on github. `https://github.com/FraudHunt`.

[eBa] eBay. `www.ebay.com`.

[EGC+10] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of OSDI*, 2010.

[EOMC11] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX Conference on Security*, 2011.

[Fac] Facebook. `https://www.facebook.com`.

[Fal14] Christos Faloutsos. Large graph mining: patterns, cascades, fraud detection, and algorithms. In *Proceedings of the 23rd international conference on World wide web*, pages 1–2. ACM, 2014.

[FCH+11] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the ACM CCS*, 2011.

[FFSG15] Shobeir Fakhraei, James Foulds, Madhusudana Shashanka, and Lise Getoor. Collective spammer detection in evolving multi-relational social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1769–1778, New York, NY, USA, 2015. ACM.

[FH16] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.

[Fiv] Fiverr. https://www.fiverr.com/.

[FLCS15] Amir Fayazi, Kyumin Lee, James Caverlee, and Anna Squicciarini. Uncovering crowdsourced manipulation of online reviews. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 233–242, New York, NY, USA, 2015. ACM.

[FML+13a] G Fei, A Mukherjee, B Liu, M Hsu, M Castellanos, and R Ghosh. Exploiting burstiness in reviews for review spammer detection. In *Proceedings of the 7th International Conference on Weblogs and Social Media, ICWSM 2013*, pages 175–184, 01 2013.

[FML+13b] Geli Fei, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Exploiting burstiness in reviews for review spammer detection. *ICWSM*, 13:175–184, 2013.

[FNG] Fake Name Generator. Your Randomly Generated Identity. https://www.fakenamegenerator.com/.

[Fre] Freelancer. http://www.freelancer.com.

[Gal90] S. I. Gallant. Perceptron-based learning algorithms. *Trans. Neur. Netw.*, 1(2):179–191, June 1990.

[GGF14] Stephan Günnemann, Nikou Günnemann, and Christos Faloutsos. Detecting anomalies in dynamic rating data: A robust probabilistic model for rating evolution. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 841–850, New York, NY, USA, 2014. ACM.

[GHP+12] Stephanie Gokhman, Jeff Hancock, Poornima Prabhu, Myle Ott, and Claire Cardie. In search of a gold standard in studies of deception. In *Proceedings of the Workshop on Computational Approaches to Deception Detection*, pages 23–30, 2012.

[GJM02] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of software engineering*. Prentice Hall PTR, 2002.

[gooa] Google Bouncer. http://goo.gl/QnC6G.

[Goob] Google Play. https://play.google.com/.

[Goo12] Google. Developer Registration. *http://goo.gl/wIwpa*, 2012.

[Gre14] Andy Greenberg. Malware Apps Spoof Android Market To Infect Phones. Forbes Security, 2014.

[Gro] Joel Grover. Don't Fall for Fake Reviews: I-Team Uncovers Them on Yelp, Facebook, Google.

[GZJS12] Michael C. Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the ACM WiSec*, 2012.

[GZZ+12] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. Riskranker: Scalable and Accurate Zero-day Android Malware Detection. In *Proceedings of ACM MobiSys*, 2012.

[HSB+15] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. BIRDNEST: bayesian inference for ratings-fraud detection. *CoRR*, abs/1511.06030, 2015.

[HSB+16a] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. Birdnest: Bayesian inference for ratings-fraud detection. In *Proceedings of SIAM SDM*, 2016.

[HSB+16b] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 895–904, New York, NY, USA, 2016. ACM.

[HSB+16c] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of ACM KDD*, 2016.

[HTGL14] Xia Hu, Jiliang Tang, Huiji Gao, and Huan Liu. Social spammer detection with sentiment information. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 180–189. IEEE, 2014.

[HTS16] Atefeh Heydari, Mohammadali Tavakoli, and Naomie Salim. Detection of fake opinions using time series. *Expert Syst. Appl.*, 58(C):83–92, October 2016.

[J1313] Google I/O 2013 - Getting Discovered on Google Play. www.youtube.com/watch?v=5Od2SuL2igA, 2013.

[JB99] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of ISOC NDSS*, 1999.

[JCB+14] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring strange behavior from connectivity pattern in social networks. In Vincent S. Tseng, Tu Bao Ho, Zhi-Hua Zhou, Arbee L. P. Chen, and Hung-Yu Kao, editors, *Advances in Knowledge Discovery and Data Mining*, pages 126–138, Cham, 2014. Springer International Publishing.

[JL07] Nitin Jindal and Bing Liu. Review spam detection. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1189–1190, New York, NY, USA, 2007. ACM.

[JSt] JStylo. The JStylo Open Source Project on Open Hub. https://www.openhub.net/p/jstylo.

[JW02] R.A. Johnson and D.W. Wichern. *Applied multivariate statistical analysis*. Prentice hall, 2002.

[Kar93] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, 1993.

[KCA17] Parisa Kaghazgaran, James Caverlee, and Majid Alfifi. Behavioral analysis of review fraud: Linking malicious crowdsourcing to amazon and beyond. In *Proceedings of ICWSM*, 2017.

[KCLS17] Srijan Kumar, Justin Cheng, Jure Leskovec, and V.S. Subrahmanian. An army of me: Sockpuppets in online discussion communities. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 857–866, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[KCS18] Parisa Kaghazgaran, James Caverlee, and Anna Squicciarini. Combating crowdsourced review manipulators: A neighborhood-based approach. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pages 306–314, New York, NY, USA, 2018. ACM.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models. Principles and Techniques.* The MIT Press, 2009.

[KM16] Santosh KC and Arjun Mukherjee. On the temporal dynamics of opinion spamming: Case studies on yelp. In *Proceedings of the 25th International Conference on World Wide Web*, pages 369–379. International World Wide Web Conferences Steering Committee, 2016.

[Koh95] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of IJCAI*, 1995.

[LAC12] Charles Z Liu, Yoris A Au, and Hoon Seok Choi. An Empirical Study of the Freemium Strategy for Mobile Apps: Evidence from the Google Play Market. In *Proceedings of ICICS*, 2012.

[LBDV97] D. Levy, M. Bergen, S. Dutta, and R. Venable. The magnitude of menu costs: direct evidence from large us supermarket chains. *The Quarterly Journal of Economics*, 112(3):791–824, 1997.

[LCM+15] Huayi Li, Zhiyuan Chen, Arjun Mukherjee, Bing Liu, and Jidong Shao. Analyzing and detecting opinion spam on a large-scale dataset via temporal and spatial patterns. In *Proceedings of ICWSM*, pages 634–637. AAAI Press, 2015.

[LCN15] Haokai Lu, James Caverlee, and Wei Niu. Biaswatch: A lightweight system for discovering and tracking topic-sensitive opinion bias in social media. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 213–222. ACM, 2015.

[LCNK17] Shanshan Li, James Caverlee, Wei Niu, and Parisa Kaghazgaran. Crowdsourced app review manipulation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1137–1140, New York, NY, USA, 2017. ACM.

[LCT12] Theodoros Lappas, Mark Crovella, and Evimaria Terzi. Selecting a characteristic set of reviews. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 832–840. ACM, 2012.

[Lev84] Henry M. Levy. *Capability-Based Computer Systems.* Butterworth-Heinemann, Newton, MA, USA, 1984.

[LFW+17a] Huayi Li, Geli Fei, Shuai Wang, Bing Liu, Weixiang Shao, Arjun Mukherjee, and Jidong Shao. Bimodal distribution and co-bursting in review spam detection. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1063–1072, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[LFW+17b] Huayi Li, Geli Fei, Shuai Wang, Bing Liu, Weixiang Shao, Arjun Mukherjee, and Jidong Shao. Bimodal distribution and co-bursting in review spam detection. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1063–1072. International World Wide Web Conferences Steering Committee, 2017.

[LLK+11] Raymond YK Lau, SY Liao, Ron Chi Wai Kwok, Kaiquan Xu, Yunqing Xia, and Yuefeng Li. Text mining and probabilistic language modeling for online review spam detecting. *ACM Transactions on Management Information Systems*, 2(4):1–30, 2011.

[LMC+16] Yixuan Li, Oscar Martinez, Xing Chen, Yi Li, and John E Hopcroft. In a world that counts: Clustering and detecting fake social engagement at scale. In *Proceedings of the 25th International Conference on World Wide Web*, pages 111–120, 2016.

[LNJ+10] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 939–948, 2010.

[Luc] Michael Luca. Reviews, Reputation, and Revenue: The Case of Yelp.com. Available at `hbswk.hbs.edu/item/6833.html`.

[Luc11] Michael Luca. Reviews, Reputation, and Revenue: The Case of Yelp.Com. *SSRN eLibrary*, 2011.

[LZ16] Michael Luca and Georgios Zervas. Fake it till you make it: Reputation, competition, and yelp review fraud. In *Management Sciences*, pages 3412–3427, 01 2016.

[LZXL13] Yuqing Lu, Lei Zhang, Yudong Xiao, and Yangguang Li. Simultaneously detecting fake reviews and review spammers using factor graph model. In *Proceedings of the ACM WebSci*, 2013.

[McD09] John H. McDonald. *Handbook of Biological Statistics*. Sparky House Publishing, second edition, 2009.

[MEY17] MICHELLE MEYERS. Fake likes: Researchers uncover Facebook 'collusion networks'. CNet, 2017.

[mic] microworkers. `https://microworkers.com/`.

[Min14] Zach Miners. Report: Malware-infected Android apps spike in the Google Play store. PCWorld, 2014.

[Mit04] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet mathematics*, 2004.

[MKL+13a] Arjun Mukherjee, Abhinav Kumar, Bing Liu, Junhui Wang, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Spotting Opinion Spammers Using Behavioral Footprints. In *Proceedings of ACM KDD*, 2013.

[MKL+13b] Arjun Mukherjee, Abhinav Kumar, Bing Liu, Junhui Wang, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Spotting opinion spammers using behavioral footprints. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 632–640. ACM, 2013.

[MLG12] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st International Conference on World Wide Web*, pages 191–200, 2012.

[Mlo14] Stephanie Mlot. Top Android App a Scam, Pulled From Google Play. PCMag, 2014.

[MMD+12] Andreas Möller, Florian Michahelles, Stefan Diewald, Luis Roalter, and Matthias Kranz. Update behavior in app markets and security implications: A case study in google play. In *Proceedings of the Intl. Workshop on Research in the Large*, 2012.

[MMF12] Fragkiskos D Malliaros, Vasileios Megalooikonomou, and Christos Faloutsos. Fast robustness estimation in large social graphs: Communities and anomaly detection. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 942–953. SIAM, 2012.

[MML+11] Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker. Dirty jobs: The role of freelance labor in web service abuse. In *Proceedings of USENIX Security*, 2011.

[MU04] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. 3111:260–272, 2004.

[MVLG13] Arjun Mukherjee, Vivek Venkataraman, Bing Liu, and Natalie Glance. What Yelp fake review filter might be doing. In *Proceedings of the AAAI ICWSM*, 2013.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[NPG+12] Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 300–314, 2012.

[OCCH11] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *Proceedings of the Human Language Technologies*, HLT '11, 2011.

[OG16] Rebekah Overdorf and Rachel Greenstadt. Blogs, twitter feeds, and reddit comments: Cross-domain authorship attribution. *PoPETs*, 2016(3), 2016.

[OK14] Judith S. Olson and Wendy A. Kellogg. *Ways of Knowing in HCI*. Springer Publishing Company, Incorporated, 2014.

[OM12] Jon Oberheide and Charlie Miller. Dissecting the Android Bouncer. *SummerCon2012, New York*, 2012.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of ACM KDD*, 2014.

[PCWF07] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: A fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 201–210, 2007.

[Peo] Peopleperhour. `https://www.peopleperhour.com/`.

[Per14] New Google Play Store greatly simplifies permissions. http://www.androidcentral.com/new-google-play-store-4820-greatly-simplifies-permissions, 2014.

[Per16] Sarah Perez. Amazon bans incentivized reviews tied to free or discounted products. Tech Crunch, https://tinyurl.com/zgn9sq3, 2016.

[PGS+12] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *Proceedings of ACM CCS*, 2012.

[PJNR13] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *Proceedings of USENIX NSDI*, 2013.

[PLV02] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs Up? Sentiment Classification Using Machine Learning Techniques. In *Proceedings of EMNLP*, 2002.

[Pos12] Huffington Post. Yelp study shows extra half-star nets restaurants 19reservations. Huffington Post, https://tinyurl.com/y7u32ssl, 2012.

[PPP+13] Thanasis Petsas, Antonis Papadogiannakis, Michalis Polychronakis, Evangelos P Markatos, and Thomas Karagiannis. Rise of the planet of the apps: A systematic study of the mobile app ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 277–290. ACM, 2013.

[PRC17] Rahul Potharaju, Mizanur Rahman, and Bogdan Carbunar. A longitudinal study of google play. *IEEE Transactions on Computational Social Systems*, 4(3):135–149, 2017.

[pri] Priceline. http://priceline.com.

[Quo] How many downloads do some of the top 50 free iphone apps get per day?

[RA15a] Shebuti Rayana and Leman Akoglu. Collective opinion spam detection: Bridging review networks and metadata. In *Proceedings of ACM KDD*, 2015.

[RA15b] Shebuti Rayana and Leman Akoglu. Collective opinion spam detection: Bridging review networks and metadata. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 985–994, New York, NY, USA, 2015. ACM.

[RA16] Shebuti Rayana and Leman Akoglu. Collective opinion spam detection using active inference. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 630–638. SIAM, 2016.

[rap] Rapidworkers. `https://rapidworkers.com/`.

[RC12] Victoria L Rubin and Niall J Conroy. The art of creating an informative data collection for automated deception detection: A corpus of truths and lies. *Proceedings of the American Society for Information Science and Technology*, 49(1):1–11, 2012.

[RL16] Rank Likes. `http://www.ranklikes.com/`, Last accessed November 2016.

[Rob15] Daniel Roberts. How to spot fake apps on the Google Play store. Fortune, 2015.

[RR16] Review Roster. `http://www.reviewroster.com/`, Last accessed November 2016.

[RRCC16a] Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar, and Duen Horng Chau. Fairplay: Fraud and malware detection in google play. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 99–107. SIAM, 2016.

[RRCC16b] Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar, and Polo Chau. Fairplay: Fraud and Malware Detection in Google Play. In *Proceedings of SDM*, 2016.

[RRCL17] Mizanur Rahman, Ruben Recabarren, Bogdan Carbunar, and Dongwon Lee. Stateless puzzles for real time online fraud preemption. In *Proceedings of the 2017 ACM on Web Science Conference*, pages 23–32. ACM, 2017.

[SBN96] Seymour Sudman, Norman M. Bradburn, and Schwarz Norbert. *Thinking about Answers: The Application of Cognitive Processes to Survey Methodology.* Jossey-Bass Publishers, 1996.

[SCM11] Tao Stein, Erdong Chen, and Karan Mangla. Facebook Immune System. In *Proceedings of the 4th Workshop on Social Network Systems*, pages 8:1–8:8, 2011.

[SE15] Vlad Sandulescu and Martin Ester. Detecting singleton review spammers using semantic similarity. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 971–976, New York, NY, USA, 2015. ACM.

[SHY+17] Chih-Ya Shen, Liang-Hao Huang, De-Nian Yang, Hong-Han Shuai, Wang-Chien Lee, and Ming-Syan Chen. On finding socially tenuous groups for online social networks. In *Proceedings of KDD*, 2017.

[Sie14] Ezra Siegel. Fake Reviews in Google Play and Apple App Store. Appentive, 2014.

[Sin15] Manish Singh. Thousands of Fraud Apps Spotted on App Store and Google Play: Report. Gadgets 360, 2015.

[SK12] Justin Sahs and Latifur Khan. A Machine Learning Approach to Android Malware Detection. In *Proceedings of EISIC*, 2012.

[SKE+12] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. Andromaly: a Behavioral Malware Detection Framework for Android Devices. *Intelligent Information Systems*, 38(1):161–190, 2012.

[SLG+12] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android Permissions: a Perspective Combining Risks and Benefits. In *Proceedings of ACM SACMAT*, 2012.

[SLK15] Jonghyuk Song, Sangho Lee, and Jong Kim. Crowdtarget: Target-based detection of crowdturfing in online social networks. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 793–804, New York, NY, USA, 2015. ACM.

[SMJ+15] Gianluca Stringhini, Pierre Mourlanne, Gregoire Jacob, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. EVILCOHORT: Detecting communities of malicious accounts on online services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 563–578, Washington, D.C., 2015. USENIX Association.

[Soc] Fraud Detection in Social Networks. https://users.cs.fiu.edu/~carbunar/caspr.lab/socialfraud.html.

[SSL+13] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pages 289–298. Springer, 2013.

[SWE+13] Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Haitao Zheng, and Ben Y. Zhao. Follow the green: Growth and dynamics in twitter follower markets. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 163–176, New York, NY, USA, 2013. ACM.

[TMG+13] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 195–210, Berkeley, CA, USA, 2013. USENIX Association.

[TRC14] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. Guilt by association: Large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1524–1533, New York, NY, USA, 2014. ACM.

[TRU] MELISSA TRUJILLO. TripAdvisor warns of hotels posting fake reviews.

[TSM16] The Social Marketeers. http://www.thesocialmarketeers.org/, Last accessed November 2016.

[Tso15] Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the Conference on World Wide Web (WWW)*, 2015.

[TTT06] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, October 2006.

[Twi] Twitter. http://www.twitter.com.

[TZX+15] Tian Tian, Jun Zhu, Fen Xia, Xin Zhuang, and Tong Zhang. Crowd fraud detection in internet advertising. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1100–1110, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[Uno07] Takeaki Uno. An efficient algorithm for enumerating pseudo cliques. In *Proceedings of ISAAC*, 2007.

[Upw] Upwork Inc. `https://www.upwork.com`.

[VGN14] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 221–233. ACM, 2014.

[Vir15] VirusTotal - Free Online Virus, Malware and URL Scanner. `https://www.virustotal.com/`, Last accessed on May 2015.

[War08] Matthijs J Warrens. On association coefficients for 2x2 tables and properties that do not depend on the marginal distributions. volume 73, 2008.

[Wei17] Elizabeth Weise. Fake Facebook 'like' networks exploited code flaw to create millions of bogus 'likes'. UsaToday, 2017.

[Wek] Weka. `http://www.cs.waikato.ac.nz/ml/weka/`.

[WGF17] Binghui Wang, Neil Zhenqiang Gong, and Hao Fu. GANG: Detecting Fraudulent Users in Online Social Networks via Guilt-by-Association on Directed Graphs. In *Proceedings of ICDM*, 2017.

[WLL+13] Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, and Wenke Lee. Jekyll on ios: When benign apps become evil. In *Proceedings of USENIX Security*, 2013.

[WSB04] E.U. Weber, S. Shafir, and A.R. Blais. Predicting risk sensitivity in humans and lower animals: risk as variance or coefficient of variation. *Psychological Review; Psychological Review*, 111(2):430, 2004.

[WWZ+12] Gang Wang, Christo Wilson, Xiaohan Zhao, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y. Zhao. Serf and Turf: Crowdturfing for Fun and Profit. In *Proceedings of ACM WWW*. ACM, 2012.

[WXLY11] Guan Wang, Sihong Xie, Bing Liu, and Philip S. Yu. Review graph based online store review spammer detection. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 1242–1247, Washington, DC, USA, 2011. IEEE Computer Society.

[WXLY12] Guan Wang, Sihong Xie, Bing Liu, and Philip S. Yu. Identify online store review spammers via social review graph. *ACM Trans. Intell. Syst. Technol.*, 3(4):61:1–61:21, September 2012.

[XEG+11] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *Proceedings of ACM IMC*, 2011.

[XFH15] Cao Xiao, David Mandell Freeman, and Theodore Hwa. Detecting clusters of fake accounts in online social networks. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 91–101. ACM, 2015.

[Xu13] Chang Xu. Detecting collusive spammers in online review communities. In *Proceedings of the Sixth Workshop on Ph.D. Students in Information and Knowledge Management*, PIKM '13, pages 33–40, New York, NY, USA, 2013. ACM.

[XWLY12] Sihong Xie, Guan Wang, Shuyang Lin, and Philip S. Yu. Review spam detection via temporal pattern discovery. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 823–831, 2012.

[XZ14] Zhen Xie and Sencun Zhu. Grouptie: Toward hidden collusion group discovery in app stores. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec '14, pages 153–164, New York, NY, USA, 2014. ACM.

[XZ15a] Zhen Xie and Sencun Zhu. Appwatcher: Unveiling the underground market of trading mobile app reviews. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 10:1–10:11, New York, NY, USA, 2015. ACM.

[XZ15b] Chang Xu and Jie Zhang. Combating product review spam campaigns via multiple heterogeneous pairwise features. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 172–180. SIAM, 2015.

[XZLW16] Zhen Xie, Sencun Zhu, Qing Li, and Wenjing Wang. You can promote, but you can't hide: Large-scale abused app detection in mobile app stores. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, pages 374–385, New York, NY, USA, 2016. ACM.

[YA15a] Junting Ye and Leman Akoglu. Discovering opinion spammer groups by network footprints. In *Machine Learning and Knowledge Discovery in Databases*. 2015.

[YA15b] Junting Ye and Leman Akoglu. Discovering opinion spammer groups by network footprints. In *Proceedings of the 2015 ACM on Conference on Online Social Networks*, COSN '15, pages 97–97, New York, NY, USA, 2015. ACM.

[Yel] Yelp. `http://www.yelp.com`.

[YFW03] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8, 2003.

[YHZ+12] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin, and Guofei Gu. Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on Twitter. In *Proceedings of the World Wide Web (WWW)*, pages 71–80. ACM, 2012.

[YKA16] Junting Ye, Santhosh Kumar, and Leman Akoglu. Temporal opinion spam detection by multivariate indicative signals. In *Proceedings of ICWSM*, pages 743–746. AAAI Press, 2016.

[YSM14] S.Y. Yerima, S. Sezer, and I. Muttik. Android Malware Detection Using Parallel Machine Learning Classifiers. In *Proceedings of NGMAST*, Sept 2014.

[YVC+17] Yuanshun Yao, Bimal Viswanath, Jenna Cryan, Haitao Zheng, and Ben Y. Zhao. Automated Crowdturfing Attacks and Defenses in Online Review Systems. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1143–1158, 2017.

[Zee] Zeerk. `https://zeerk.com/`.

[ZJ12] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the IEEE S&P*, pages 95–109. IEEE, 2012.

[ZL15] Reza Zafarani and Huan Liu. 10 bits of surprise: Detecting malicious users with minimum information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 423–431, 2015.

[ZM13] Nan Zhong and Florian Michahelles. Google play is not a long tail market: An empirical analysis of app adoption on the google play app market. In *Proceedings of the ACM SAC*, 2013.

[ZWZJ12] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of NDSS*, 2012.

[ZXGC13] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Ranking fraud detection for mobile apps: A holistic view. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 619–628, New York, NY, USA, 2013. ACM.

[ZXL+18] Haizhong Zheng, Minhui Xue, Hao Lu, Shuang Hao, Haojin Zhu, Xiaohui Liang, and Keith Ross. Smoke Screener or Straight Shooter: Detecting Elite Sybil Attacks in User-Review Social Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

VITA

MD MIZANUR RAHMAN

2013 - Now	Ph.D., Computer Science Florida International University Miami, Florida
2009 - 2013	Software Engineer KAZ Software Dhaka, Bangladesh
2003 - 2008	B.Sc., Computer Science Bangladesh Univ of Eng and Tech Dhaka, Bangladesh

PUBLICATIONS

1. M. Rahman, N. Hernandez, B. Carbunar and D. Chau. *Search Rank Fraud De-Anonymization in Online Systems*. The 29th ACM Conference on Hypertext and Social Media (HT), 2018.
2. M. Rahman, N. Hernandez, B. Carbunar and D. Chau. *DOLOS: Towards De-Anonymization of Search Rank Fraud*. In IEEE Transactions on Knowledge and Data Engineering (TKDE) [Submitted]
3. N. Hernandez, M. Rahman, R. Recabarren, B. Carbunar. *Fraud De-Anonymization For Fun and Profit*. The 25th ACM Conference on Computer and Communications Security (CCS), 2018.
4. M. Rahman, R. Recabarren, B. Carbunar, and D. Lee. *Stateless puzzles for real time online fraud preemption*. In Proceedings of the 9th ACM Web Science Conference (WebSci), September 2017.
5. R. Potharaju, M. Rahman, B. Carbunar *A Longitudinal Study of Google Play*. In IEEE Transactions on Computational Social Systems (TCSS), Volume 4, Issue 3, September 2017.
6. M. Rahman, M. Rahman, B. Carbunar, D. H. Chau. *Search Rank Fraud and Malware Detection in Google Play*. In IEEE Transactions on Knowledge and Data Engineering (TKDE), Volume 29, Issue 6, June 2017.
7. M. Rahman, M. Rahman, B. Carbunar, D. H. Chau. *FairPlay: Fraud and Malware Detection in Google Play*. In Proceedings of the SIAM International Conference on Data Mining (SDM), May 2016.
8. B. Carbunar, M. Rahman, M. Azimpourkivi, D. Davis. *GeoPal: Friend Spam Detection in Social Networks Using Private Location Proofs* in IEEE SECON, 2016.

9. M. A. Sharmin, M. Rahman, S. I. Ahmed, M. M. Rahman, S. M. Ferdous, *Teaching intelligible speech to the Autistic children by interactive computer games*, ACM SAC, Taiwan, China, 2011.

10. I. Ahmed, S. Samrose, S. Naha, M. R. Rahman, P. C. Roy, M. Rahman, S. I. Ahmed, *A-Class: Intelligent Classroom Software for the Autistic Children* , IEEE ISCI, Penang, Malaysia, March 2011.