FairPlay: Fraud and Malware Detection in Google Play

Mahmudur Rahman
Florida Int'l Univ.
mrahm004@fiu.edu

Mizanur Rahman
Florida Int'l Univ.
mrahm031@fiu.edu

Bogdan Carbunar
Florida Int'l Univ.
carbunar@gmail.com

Duen Horng Chau
Georgia Tech
polo@gatech.edu

Abstract

Fraudulent behaviors in Google's Android app market fuel search rank abuse and malware proliferation. We present FairPlay, a novel system that uncovers both malware and search rank fraud apps, by picking out trails that fraudsters leave behind. To identify suspicious apps, FairPlay's PCF algorithm correlates review activities and uniquely combines detected review relations with linguistic and behavioral signals gleaned from longitudinal Google Play app data. We contribute a new longitudinal app dataset to the community, which consists of over 87K apps, 2.9M reviews, and 2.4M reviewers, collected over half a year. FairPlay achieves over 95% accuracy in classifying gold standard datasets of malware, fraudulent and legitimate apps. We show that 75% of the identified malware apps engage in search rank fraud. FairPlay discovers hundreds of fraudulent apps that currently evade Google Bouncer's detection technology, and reveals a new type of attack campaign, where users are harassed into writing positive reviews, and install and review other apps.

1 Introduction

The commercial success of Android app markets such as Google Play [1] has made them a lucrative medium for committing fraud and malice. Some fraudulent developers deceptively boost the search ranks and popularity of their apps (e.g., through fake reviews and bogus installation counts) [2], while malicious developers use app markets as a launch pad for their malware [3, 4, 5, 6].

Existing mobile malware detection solutions have limitations. For instance, while Google Play uses the Bouncer system [7] to remove malware, out of the 7,756 Google Play apps we analyzed using VirusTotal [8], 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 3a). Previous work has focused on dynamic analysis of app executables [9, 10, 11] as well as static analysis of code and permissions [12, 13, 14]. However, recent Android malware analysis revealed that malware evolves quickly to bypass anti-virus tools [15].

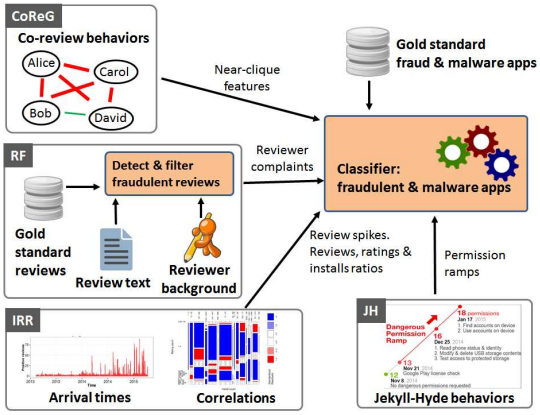


Figure 1: FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.

In this paper, we seek to identify both malware and search rank fraud targets 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 our 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. Ramps in the number of "dangerous" permissions requested by apps may indicate benign to malware (Jekyll-Hyde) transitions.

Contributions and Results. We propose FairPlay, a system that leverages the above observations to efficiently detect Google Play fraud and malware (see Fig-

ure 1). Our major contributions are:

• **A unified relational, linguistic and behavioral approach**. 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 linguistic and behavioral information to (i) detect genuine reviews from which we then (ii) extract user-identified fraud and malware indicators. In addition, we detect apps with (i) permission request ramps, (ii) "unbalanced" review, rating and install counts, and (iii) suspicious review spikes. We generate 28 features, and use them to train supervised learning algorithms [§ 4].

• **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 [16], anti-virus tools and manual verifications to collect gold standard datasets of hundreds of fraudulent, malware and benign apps. We will publish these datasets alongside this work [§ 3].

• **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. [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 [§ 5.3].

• **Real-world Impact: Uncover Fraud & Attacks**. FairPlay discovers hundreds of fraudulent apps that currently evade Google Bouncer's detection technology. We show that these apps are indeed suspicious: the reviewers of 93.3% of them form at least 1 pseudo clique and 55% of these apps have at least 33% of their reviewers involved in a pseudo clique. In addition, FairPlay enabled us to discover a novel, *coercive campaign* attack type, where app users are harassed into writing a positive review for the app, and install and review other apps [§ 5.4 & § 5.5].

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 (1-5 stars rating & text), ratings (1-5 stars, no text), 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.

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. 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., [17]. Fraudulent developers often rely on crowdsourcing sites [16, 18, 19] to hire teams of workers to commit fraud collectively.

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 review writing jobs.

2.1 Research in Android Malware Detection.

Burguera et al. [9] 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. [10] extracted features from monitored apps (e.g., CPU consumption, packets sent, running processes) and used machine learning to identify malicious apps. Grace et al. [11] used static analysis to efficiently identify high and medium risk apps.

Previous work has also used app permissions to pinpoint malware [12, 13, 14]. Sarma et al. [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. In § 5.3 we use Sarma et al. [12]'s solution as a baseline, and show that FairPlay significantly improves on its performance.

Peng et al. [13] propose a score to measure the risk of apps, based on probabilistic generative models such as Naive Bayes. Yerima et al. [14] also use features extracted from app permissions, API calls and commands extracted from the app executables.

Instead of analyzing app executables, FairPlay employs a unified 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.

2.2 Research on Graph Based Opinion Spam Detection. Graph based approaches have been proposed to tackle opinion spam [20, 21]. Ye and Akoglu [20] 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. [21] frame the 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.

3 The Data

We have collected longitudinal data from 87K+ newly released apps over more than 6 months, and identified gold standard app market behaviors. 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 shows only 20 apps on a user page by default. GPCrawler overcomes this limitation by using a Firefox add-on and a Python script. The add-on interacts with Google Play to extend the user page with a "scroll down" button and enable the script to automatically navigate and collect all the information from the user page.

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* [22]. GPad scans each app apk using VirusTotal [8], 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.

3.1 Longitudinal App Data. In order to detect app attribute 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. 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.

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 § 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.

3.2 Gold Standard Data.

Malware apps. We used GPad (see § 3) to collect the apks of $7,756$ randomly selected apps from the longitudinal set (see § 3.1). Figure 3a shows the distribution of flags raised by VirusTotal, for the $7,756$ apks. None of these apps had been filtered by Bouncer [7]! 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.

Fraudulent apps. We used contacts established among Freelancer [16]'s search rank fraud community, to obtain the identities of 15 Google Play accounts that were used to write fraudulent reviews. We call these "seed fraud accounts". These accounts were used to review 201 unique apps. We call these, the "seed fraud apps", and we use them to evaluate FairPlay.

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. 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 standard dataset of fraudulent reviews consists of 406 reviews.

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

Notation	Definition
CoReG Module	
$nCliques$	number of pseudo cliques with $\rho \geq \theta$
stats(ρ)	clique density: max, median, SD
stats($cliqueSize$)	pseudo cliques size: max, median, SD
$inCliqueSize$	% 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	
stats($spikes$)	days with spikes & spike amplitude
I_1/Rt_1, I_2/Rt_2	install to rating ratios
I_1/Rt_1, I_2/Rt_2	install to review ratios
JH Module	
$permCt$, $dangerCt$	# of dangerous and total permissions
$rampCt$	# of dangerous permission ramps
$dangerRamp$	# of dangerous permissions added

Table 1: FairPlay's most important features, organized by their extracting module.

10 reviews and (ii) were developed by reputable media outlets (e.g., NBC, PBS) or have an associated business model (e.g., fitness trackers).

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 FairPlay: Proposed Solution

4.1 FairPlay Overview. FairPlay organizes the analysis of longitudinal app data into the following 4 modules, illustrated in Figure 1. 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,

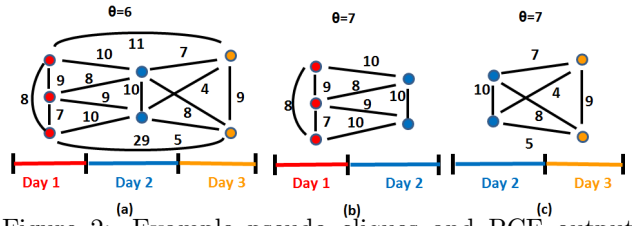


Figure 2: Example pseudo cliques and PCF output. Nodes are users and edge weights denote the number of apps reviewed in common by the end users. Review timestamps have a 1-day granularity. (a) The entire co-review graph, detected as pseudo clique by PCF when θ is 6. When θ is 7, PCF detects the subgraphs of (b) the first two days and (c) the last two days.

ratings and installs, for a total of 28 features. Table 1 summarizes the most important features. In the following, we detail each module and the features it extracts.

4.2 The Co-Review Graph (CoReG) Module. Let the *co-review graph* of an app, see Figure 2, be a graph where nodes correspond to users 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. We seek to identify cliques in the co-review graph. Figure 5a shows the co-review clique of one of the seed fraud apps (see § 3.2).

To address the problem's NP-hardness, we exploit two observations. First, fraudsters hired to review an app are likely to post those reviews within relatively short time intervals (e.g., days). Second, perfect cliques are not necessary. Instead, we relax this requirement to identify "pseudo cliques", or groups of highly but not necessarily completely connected nodes. Specifically, we use the weighted density definition of Uno [23]: given a co-review graph, its weighted density $\rho = \frac{\sum_{e \in E} w(e)}{\binom{n}{2}}$, where E denotes the graph's edges and n its number of nodes (reviews). We are interested then in subgraphs of the co-review graph whose weighted density exceeds a threshold value θ.

We present the Pseudo Clique Finder (PCF) algorithm (see Algorithm 1), that 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 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 on to the next day (line 5): greedily add other reviews while the weighted density of the new pseudo

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 (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
```

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 2 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 $O(dr^2(r+d))$.

CoReG features. CoReG extracts the following features from the output of PCF (see Table 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.

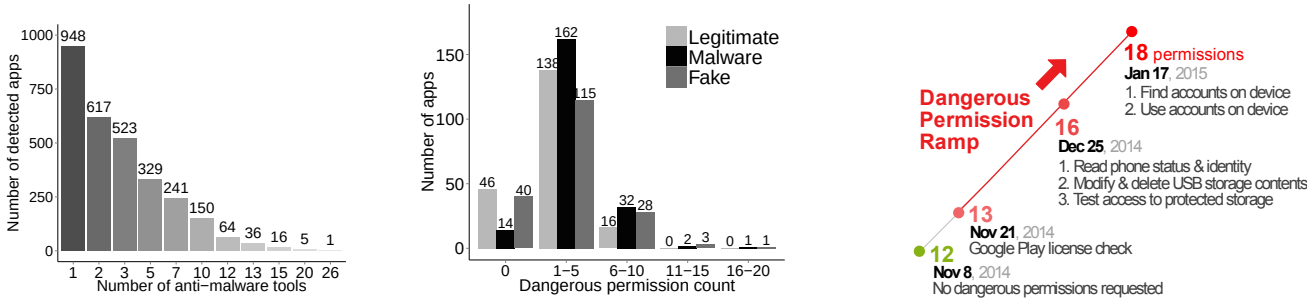
4.3 Reviewer Feedback (RF) Module. Reviews written by genuine users of malware and fraudulent apps may describe negative experiences. The RF module exploits this observation through a two step approach: (i) detect and filter out fraudulent reviews, then (ii) identify malware and fraud indicative feedback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

Step RF.1: Fraudulent review filter. We posit that users that have higher expertise on apps they review, have written fewer reviews for apps developed by the same developer, have reviewed more paid apps, are more likely to be genuine. We exploit this conjecture to use supervised learning algorithms trained on the following features, 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 § 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 [24] 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 [25]. 10-fold cross validation of the Naive Bayes classifier over these datasets reveals a FNR of 16.1% and FPR of 19.65%. 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.

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 § 3.2). We used these words to manually identify lists of words indicative of malware, fraudulent and benign behaviors. Our malware indicator word list



(a) (b) (c)

Figure 3: (a) Apks detected as suspicious (y axis) by multiple anti-virus tools (x axis), through VirusTotal [8], from a set of 7,756 downloaded apks. (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.

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 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 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.

Temporal relations. We detect outliers in the number of daily reviews received by an app. 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.

Reviews, ratings and install counts. We used the Pearson's χ^2 test to investigate relationships between the install and rating counts 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 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. We leverage this result to conjecture that adversaries that post fraudulent ratings and reviews, or create fake app install events, may break a natural balance between their counts.

IRR features. We extract temporal features (see Table 1): the number of days with detected spikes and

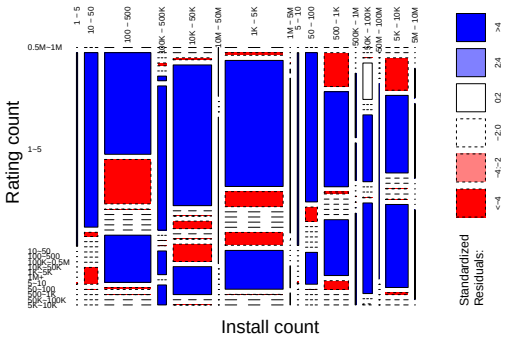


Figure 4: Mosaic plot of install vs. rating count relations of the 87K apps. Larger rectangles signify that more apps have the corresponding rating and install count range; dotted lines mean no apps in a certain install/rating category. The standardized residuals identify the cells that contribute the most to the χ^2 test. The most significant rating:install ratio is 1:100.

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.

4.5 Jekyll-Hyde App Detection (JH) Module.

Android's API level 22 labels 47 permissions as "dangerous". Figure 3b 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". Most benign apps request at most 5 such permissions; some malware and fraudulent apps request more than 10.

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 2: 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%.

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 3c shows the dangerous permissions added during different version updates of one gold standard malware app.

JH features. We extract the following features (see Table 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.

5 Evaluation

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 [26] 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) [27], Decision Trees (DT) (C4.5) and Random Forest (RF) [28], using 10-fold cross-validation [29]. 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*.

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 § 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). Table 2 shows the results of the 10-fold cross validation of algorithms classifying reviews as genuine or fraudulent. To minimize wrongful accusations, we seek to minimize the FPR [30]. MLP simultaneously achieves the highest accuracy of 96.26% and the lowest FPR of 1.47% (at 6.67% FNR). 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 3: FairPlay classification results (10-fold cross validation) of gold standard fraudulent (positive) and benign apps. RF has lowest FPR, thus desirable [30].

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. [12]/SVM	65.32	24.47	55.23

Table 4: FairPlay classification results (10-fold cross validation) of gold standard malware (positive) and benign apps, significantly outperforming Sarma et al. [12]. FairPlay's RF achieves 96.11% accuracy at 1.51% FPR.

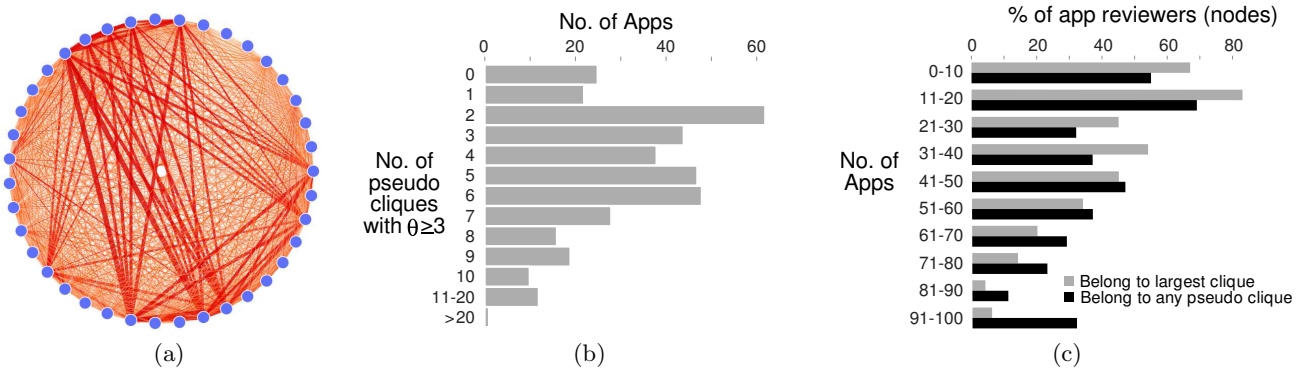
5.3 App Classification

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 or played by these users.

Fraud Detection Accuracy. Table 3 shows 10-fold cross validation results of FairPlay on the gold standard fraudulent and benign apps (see § 3.2). All classifiers achieve accuracies of around 97%. Random Forest is the best, having the highest accuracy of 97.74% and the lowest FPR of 1.01%.

Figure 5a shows the co-review subgraph for one of the seed fraud apps identified by FairPlay's PCF. We observe that the app's reviewers form a tightly connected clique, with any two reviewers having reviewed at least 115 and at most 164 apps in common.

Malware Detection Accuracy. We have used Sarma et al. [12]'s solution as a baseline to evaluate the ability of FairPlay to accurately detect malware. We computed Sarma et al. [12]'s RCP and RPCP indicators (see § 2.1) using the longitudinal app dataset. We used the SVM based variant of Sarma et al. [12], which performs best. Table 3 shows 10-cross validation results over the malware and benign gold standard sets. FairPlay significantly outperforms Sarma et al. [12]'s



(a) (b) (c)

Figure 5: (a) Clique flagged by PCF for "Tiempo - Clima gratis", one of the 201 seed fraud apps (see § 3.2), involving 37 reviewers (names hidden for privacy); edge weights proportional to numbers of apps reviewed in common (ranging from 115 to 164 apps). (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!

solution, with an accuracy that consistently exceeds 95%. Random Forest has the smallest FPR of 1.51% and the highest accuracy of 96.11%. 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.

5.4 FairPlay on the Field. We have also evaluated FairPlay on non "gold standard" apps. For this, we have collected a set of apps, as follows. First, we selected 8 app categories: Arcade, Entertainment, Photography, Simulation, Racing, Sports, Lifestyle, Casual. We have 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.

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. During the 10-fold cross validation of FairPlay for the gold standard fraudulent and benign sets, the top most impactful features for the Decision Tree classifier were (i) the percentage of nodes that belong to the largest pseudo clique, (ii) the percentage of nodes that belong to at least one pseudo clique, (iii) the percentage of reviews that contain fraud indicator words, and (iv) the number of pseudo clique with $\theta \geq 3$.

We use these features to offer an intuition for the surprisingly high fraud percentage (23% of 1,600 apps). Figure 5b 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 5c 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 single pseudo clique containing 33% of their reviewers. While not plotted here due to space constraints, we note that around 75% of the 372 fraudulent apps have at least 20 fraud indicator words in their reviews.

5.5 Coercive Campaign Apps. Upon close inspection of apps flagged as fraudulent by FairPlay, we identified apps perpetrating a new attack type. The apps, which we call *coercive campaign apps*, 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). In return, the app rewards the user by, e.g., removing ads, providing more features, unlocking the next game level, boosting the user's game level or awarding game points.

We found evidence of coercive campaign apps from users complaining through reviews, e.g., "I only rated it because 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".

We leveraged this evidence to identify more coercive campaign apps from the longitudinal app set. Specifically, we have first manually selected a list of potential keywords indicating coercive apps (e.g., "rate", "download", "ads"). We then searched all the $2,850,705$ reviews of the 87K apps and found around 82K reviews that contain at least one of these keywords. Due to time constraints, we then randomly selected $3,000$ reviews from this set, that are not flagged as fraudulent by FairPlay's RF module. Upon manual inspection, we identified 118 reviews that report coercive apps, and 48 apps that have received at least 2 such reviews. We leave a more thorough investigation of this phenomenon for future work.

6 Conclusions

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.

7 Acknowledgments

This research was supported in part by NSF grants 1527153 and 1526254, and DoD W911NF-13-1-0142.

References

[1] Google Play. `https://play.google.com/`.

[2] Ezra Siegel. Fake Reviews in Google Play and Apple App Store. Appentive, 2014.

[3] Zach Miners. Report: Malware-infected Android apps spike in the Google Play store. PCWorld, 2014.

[4] Stephanie Mlot. Top Android App a Scam, Pulled From Google Play. PCMag, 2014.

[5] Daniel Roberts. How to spot fake apps on the Google Play store. Fortune, 2015.

[6] Andy Greenberg. Malware Apps Spoof Android Market To Infect Phones. Forbes Security, 2014.

[7] Jon Oberheide and Charlie Miller. Dissecting the Android Bouncer. *SummerCon2012, New York*, 2012.

[8] VirusTotal - Free Online Virus, Malware and URL Scanner. `https://www.virustotal.com/`, Last accessed on May 2015.

[9] 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.

[10] 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.

[11] 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.

[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.

[13] 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.

[14] S.Y. Yerima, S. Sezer, and I. Muttik. Android Malware Detection Using Parallel Machine Learning Classifiers. In *Proceedings of NGMAST*, Sept 2014.

[15] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the IEEE S&P*, pages 95–109. IEEE, 2012.

[16] Freelancer. `http://www.freelancer.com`.

[17] Google I/O 2013 - Getting Discovered on Google Play. `www.youtube.com/watch?v=5Od2SuL2igA`, 2013.

[18] Fiverr. `https://www.fiverr.com/`.

[19] BestAppPromotion. `www.bestreviewapp.com/`.

[20] Junting Ye and Leman Akoglu. Discovering opinion spammer groups by network footprints. In *Machine Learning and Knowledge Discovery in Databases*, pages 267–282. Springer, 2015.

[21] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion Fraud Detection in Online Reviews by Network Effects. In *Proceedings of ICWSM*, 2013.

[22] Android Market API. `https://code.google.com/p/android-market-api/`, 2011.

[23] Takeaki Uno. An efficient algorithm for enumerating pseudo cliques. In *Proceedings of ISAAC*, 2007.

[24] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly, 2009.

[25] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs Up? Sentiment Classification Using Machine Learning Techniques. In *Proceedings of EMNLP*, 2002.

[26] Weka. `http://www.cs.waikato.ac.nz/ml/weka/`.

[27] S. I. Gallant. Perceptron-based learning algorithms. *Trans. Neur. Netw.*, 1(2):179–191, June 1990.

[28] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[29] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of IJCAI*, 1995.

[30] 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.