Search Rank Fraud and Malware Detection in Google Play

Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar, and Duen Horng Chau

Abstract—Fraudulent behaviors in Google Play, the most popular Android app market, fuel search rank abuse and malware proliferation. To identify malware, previous work has focused on app executable and permission analysis. In this paper, we introduce FairPlay, a novel system that discovers and leverages traces left behind by fraudsters, to detect both malware and apps subjected to search rank fraud. FairPlay correlates review activities and uniquely combines detected review relations with linguistic and behavioral signals gleaned from Google Play app data (87 K apps, 2.9 M reviews, and 2.4M reviewers, collected over half a year), in order to identify suspicious apps. FairPlay achieves over 95 percent accuracy in classifying gold standard datasets of malware, fraudulent and legitimate apps. We show that 75 percent of the identified malware apps engage in search rank fraud. FairPlay discovers hundreds of fraudulent apps that currently evade Google Bouncer's detection technology. FairPlay also helped the discovery of 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.

Index Terms—Android market, search rank fraud, malware detection

1 Introduction

The commercial success of Android app markets such as Google Play [1] 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) [2], while malicious developers use app markets as a launch pad for their malware [3], [4], [5], [6]. 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 [7], Fiverr [8], BestAppPromotion [9]) to hire teams of willing workers to commit fraud collectively, emulating realistic, spontaneous activities from unrelated people (i.e., "crowdturfing" [10]), see Fig. 1 for an example. We call this behavior "search rank fraud".

In addition, the efforts of Android markets to identify and remove malware are not always successful. For instance, Google Play uses the Bouncer system [11] to remove malware. However, out of the 7,756 Google Play apps we analyzed using VirusTotal [12], 12 percent (948) were flagged by at least one anti-virus tool and 2 percent (150) were identified as malware by at least 10 tools (see Fig. 6).

- *M. Rahman is with IBM, Raleigh, NC 27703. E-mail: mrahman.fiu@gmail.com.*
- *M. Rahman and B. Carbunar are with FIU, Miami, FL 33199. E-mail: {mrahm031, carbunar}@cs.fiu.edu.*
- *D.H. Chau is with Georgia Tech, Atlanta, GA 30332. E-mail: polo@gatech.edu.*

Manuscript received 1 June 2016; revised 17 Dec. 2016; accepted 4 Feb. 2017. Date of publication 13 Feb. 2017; date of current version 27 Apr. 2017. Recommended for acceptance by L. Akoglu. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2017.2667658

Previous mobile malware detection work has focused on dynamic analysis of app executables [13], [14], [15] as well as static analysis of code and permissions [16], [17], [18]. However, recent Android malware analysis revealed that malware evolves quickly to bypass anti-virus tools [19].

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.

1.1 Contributions

We propose FairPlay, a system that leverages the above observations to efficiently detect Google Play fraud and malware (see Fig. 7). 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 [Section 4]:

- We formulate the notion of *co-review graphs* to model reviewing relations between users. We develop PCF,

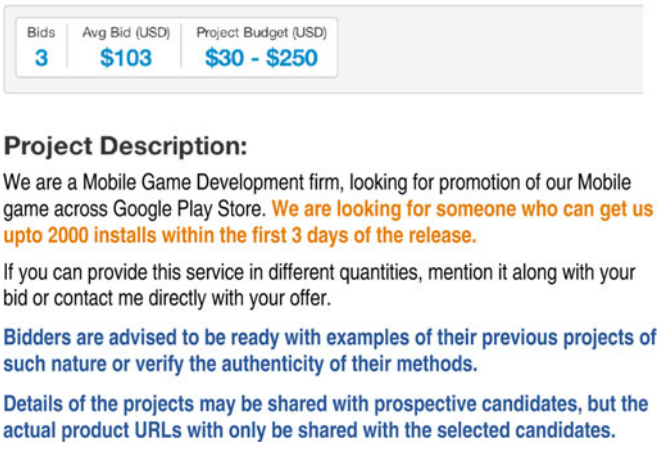

Fig. 1. An "install job" posting from Freelancer [7], asking for 2,000 installs within 3 days (in orange), in an organized way that includes expertise verifications and provides secrecy assurances (in blue). Text enlarged for easier reading.

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 2.9 M reviews, from 2.3 M reviewers) collected between October 2014 and May 2015. We have leveraged search rank fraud expert contacts in Freelancer [7], anti-virus tools and manual verifications to collect gold standard datasets of hundreds of fraudulent, malware and benign apps [§ 3]. We have published these datasets on the project website [20].

1.2 Results

FairPlay has high accuracy and real-world impact:

High Accuracy. FairPlay achieves over 97 percent accuracy in classifying fraudulent and benign apps, and over 95 percent accuracy in classifying malware and benign apps. FairPlay significantly outperforms the malware indicators of Sarma et al. [16]. 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 percent of the gold standard malware apps section 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 93.3 percent of them form at least 1 pseudo-clique, 55 percent of these apps have at least 33 percent of their reviewers involved in a pseudo-clique, and the reviews of around 75 percent of these apps contain at least 20 words indicative of fraud.

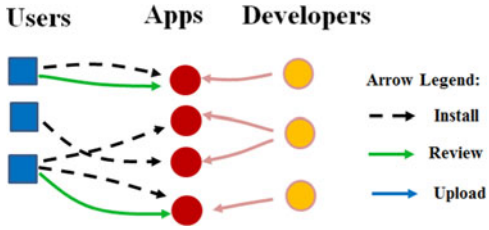

Fig. 2. Google Play components and relations. Google Play's functionality centers on apps, shown as red disks. Developers, shown as orange disks upload apps. A developer may upload multiple apps. Users, shown as blue squares, can install and review apps. A user can only review an app that he previously installed.

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 [Sections 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, 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. Fig. 2 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., [21]).

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.

2.1 Android Malware Detection

Zhou and Jiang [19] 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. [13] 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. [14] extracted features from monitored apps (e.g., CPU consumption, packets sent, running processes) and used machine learning to identify malicious apps. Grace et al. [15] used static analysis to efficiently identify high and medium risk apps.

Previous work has also used app permissions to pinpoint malware [16], [17], [18]. Sarma et al. [16] 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 versus benefits tradeoffs of apps. In Section 5.3 we show that FairPlay significantly improves on the performance achieved by Sarma et al. [16].

Peng et al. [17] propose a score to measure the risk of apps, based on probabilistic generative models such as Naive Bayes. Yerima et al. [18] also use features extracted from app permissions, API calls and commands extracted from the app executables.

Sahs and Khan [22] used features extracted from app permissions and control flow graphs to train an SVM classifier on 2,000 benign and less than 100 malicious apps. Sanz et al. [23] 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 [11] 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.

2.2 Graph Based Opinion Spam Detection

Graph based approaches have been proposed to tackle opinion spam [24], [25]. Ye and Akoglu [24] 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. [25] 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.

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* [26]. 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 [12], 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 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

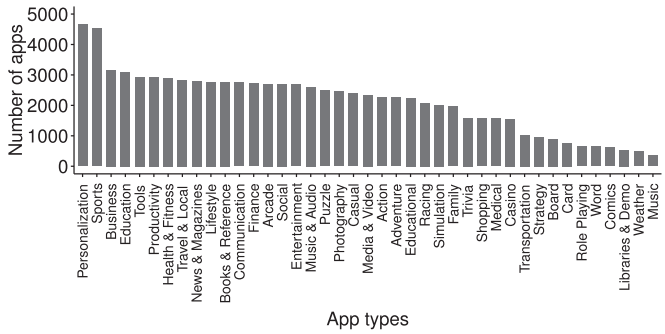

Fig. 3. Distribution of app types for the 87,223 fresh app set. With the slight exception of "Personalization" and "Sports" type spikes, we have achieved an almost uniform distribution across all app types, as desirable.

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.

Fig. 3 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 (Personalization). Fig. 4 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

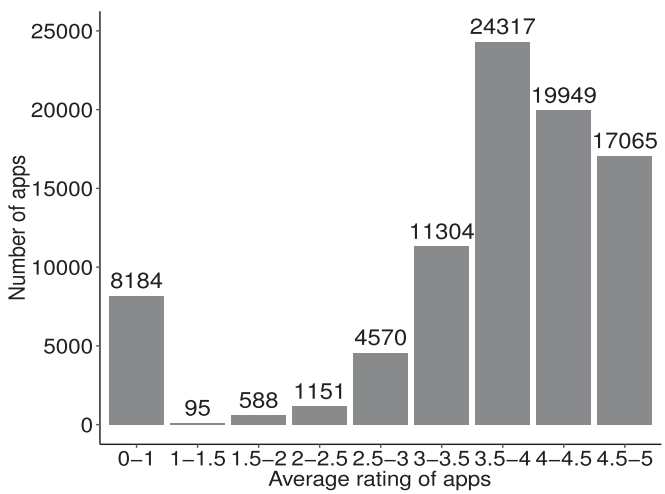

Fig. 4. Average rating distribution for the 87,223 fresh app set. Most apps have more than 3.5 stars, few have between 1 and 2.5 stars, but more than 8,000 apps have less than 1.

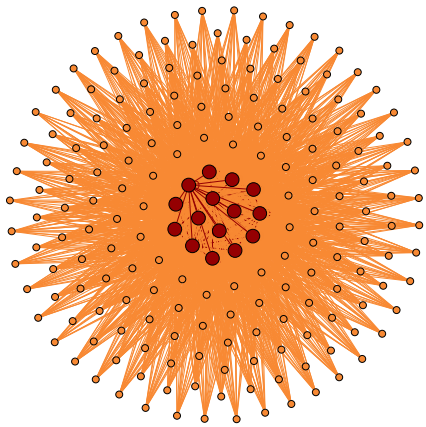

Fig. 5. Co-review graph of 15 seed fraud accounts (red nodes) and the 188 GbA accounts (orange nodes). Edges indicate reviews written in common by the accounts corresponding to the endpoints. We only show edges having at least one seed fraud account as an endpoint. The 15 seed fraud accounts form a suspicious 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.

permissions (see Section 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 4,000 displayed reviews per app: each snapshot will capture only the reviews posted after the previous snapshot.

3.2 Gold Standard Data

Malware Apps. We used GPad (see Section 3) to collect the apks of 7,756 randomly selected apps from the longitudinal set (see Section 3.1). Fig. 6 shows the distribution of flags raised by VirusTotal, for the 7,756 apks. None of these apps had been filtered by Bouncer [11]! 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 [7]'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". Fig. 5 shows the graph formed by the review habits of the 15 seed accounts: nodes are accounts, edges connect 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

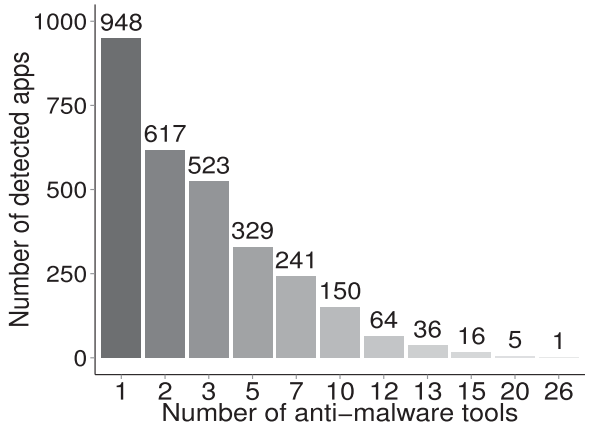

Fig. 6. Apks detected as suspicious (y axis) by multiple anti-virus tools (x axis), through VirusTotal [12], from a set of 7,756 downloaded apks.

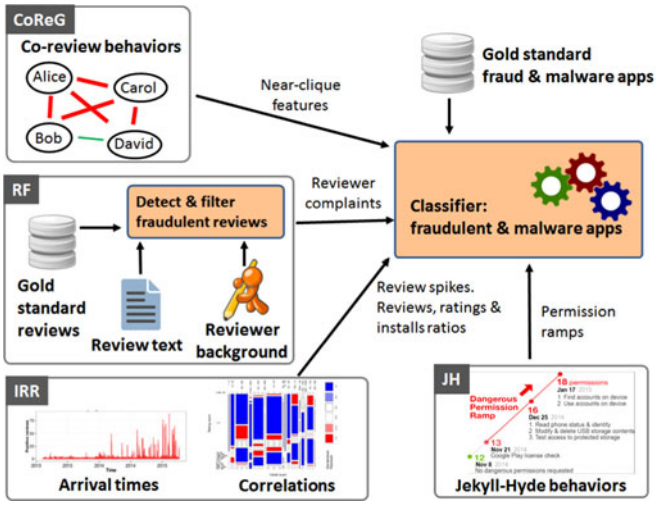

Fig. 7. FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.

201 seed fraud apps (for a total of 6,488 reviews). We call these, *guilt by association* (GbA) accounts. Fig. 5 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 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 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 FairPlay: Proposed Solution

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

4.1 FairPlay Overview

FairPlay organizes the analysis of longitudinal app data into the following 4 modules, illustrated in Fig. 7. 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 1

TABLE 1
FairPlay's Most Important Features, Organized by Their Extracting Module

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

Section 4.2 describes ρ and θ.

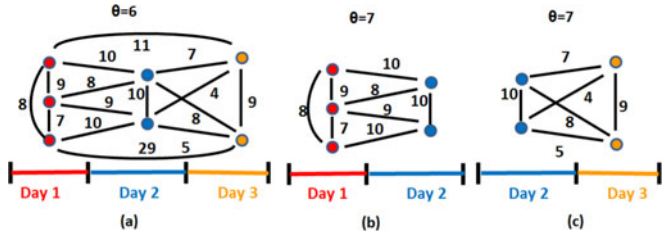

Fig. 8. Example pseudo-cliques and PCF output. Nodes are users and edge weights denote the number of apps reviewed in common by the end users. Review timestamps have a 1-day granularity. (a) The entire co-review graph, detected as pseudo-clique by PCF when θ is 6. When θ is 7, PCF detects the subgraphs 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).

summarizes the most important features. We now detail each module and the features it extracts.

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 Fig. 8, 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. Fig. 16a shows the co-review clique of one of the seed fraud apps (see Section 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.

Maximal clique enumeration algorithms such as [27], [28] 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 $\rho = \frac{\sum_{e \in E} w(e)}{\binom{n}{2}}$ [29] 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 5.3 we discuss the choice of θ.

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
```

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-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)

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.

TABLE 2
Features Used to Classify Review R Written by User U for App A

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+

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. Fig. 8 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 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 certain features can accurately pinpoint genuine and fake reviews. We propose several such features, see Table 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 Section 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 [30] 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 [31]. 10-fold cross validation of the Naive Bayes classifier over these datasets reveals a false negative rate of 16.1 percent and a false positive rate of 19.65 percent, for an overall accuracy of 81.74 percent. We ran a binomial test [32] for a given accuracy of p=0.817 over N=1,041 cases using the binomial distribution $binomial(p, N)$ to assess the 95 percent confidence interval for our result. The deviation of the binomial distribution is 0.011. Thus, we are 95 percent 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 5 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 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 Section 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 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. In order to compensate for a negative review, an attacker needs to post a significant number of positive reviews. Specifically,

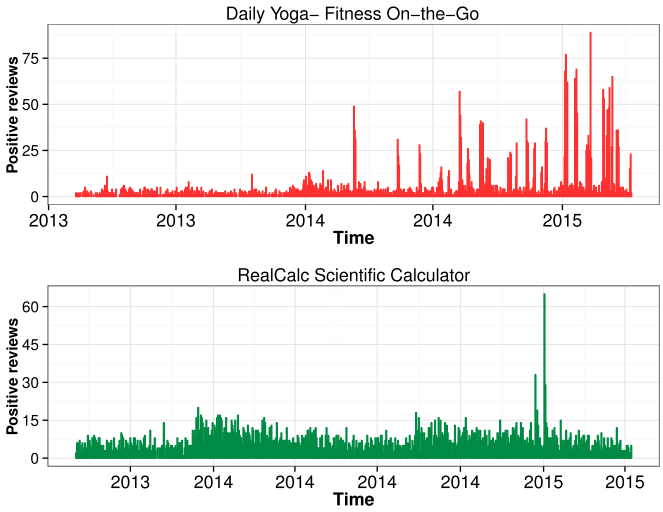

Fig. 9. Timelines of positive reviews for 2 apps from the fraudulent app dataset. The first app has multiple spikes while the second one has only one significant spike.

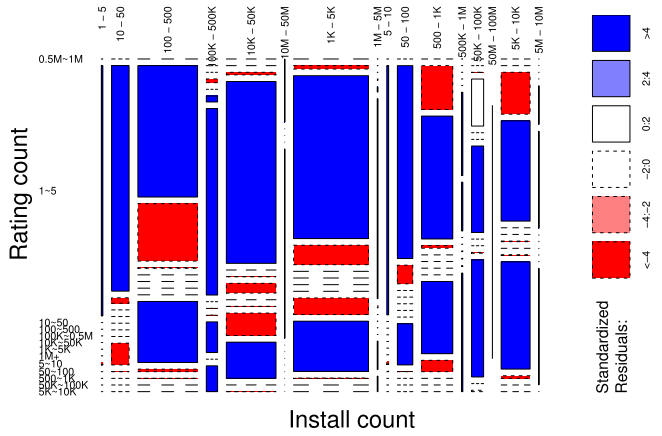

Fig. 10. Mosaic plot of install versus rating count relations of the 87K apps. Larger cells (rectangles) signify that more apps have the corresponding rating and install count range; dotted lines mean no apps in a certain install/rating category. The standardized residuals identify the cells that contribute the most to the χ^2 test. The most significant rating: install ratio is 1:100.

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. □

Fig. 13 plots the lower bound on the number of fake reviews that need to be posted to cancel a 1-star review, versus 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 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. Fig. 9 shows the timelines and suspicious spikes of positive reviews for 2 apps from the fraudulent app dataset (see Section 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.

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 87 K 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. Fig. 10 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, Fig. 11 shows the mosaic plot of the app install count versus 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),

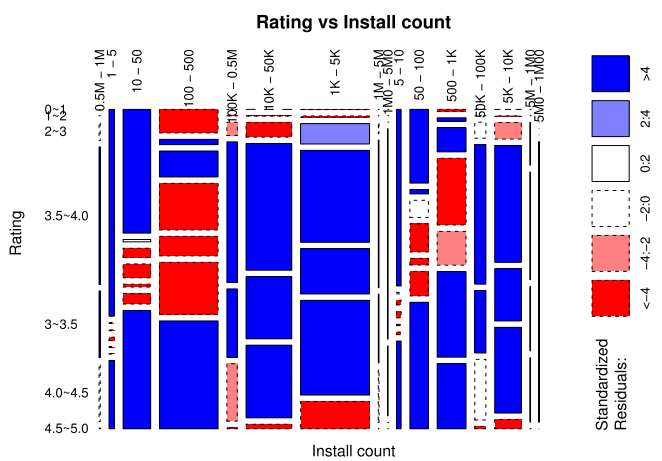

Fig. 11. Mosaic plot showing relationships between the install count and the average app rating, over the 87K apps. A cell contains the apps that have a certain install count interval (x axis) and rating interval (y axis). Larger cells contain more apps. We observe a relationship between install count 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.

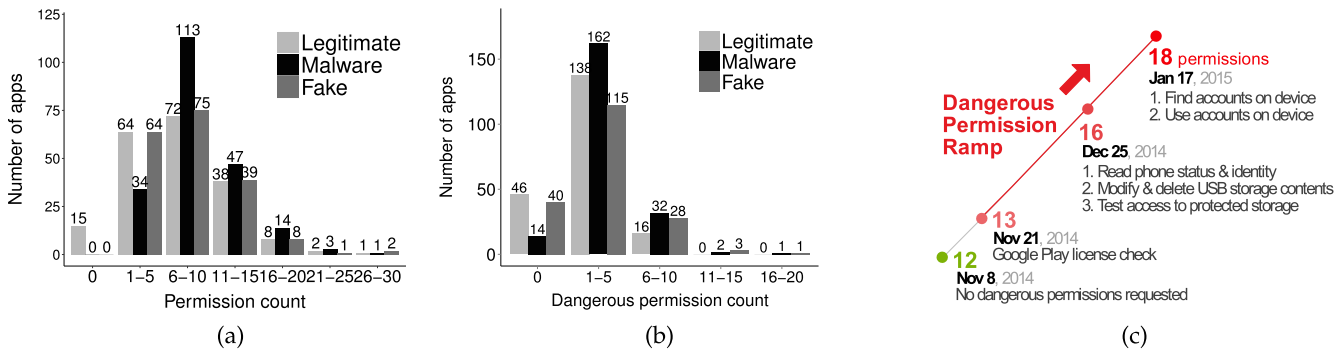

Fig. 12. (a) Distribution of total number of permissions requested by malware, fraudulent and legitimate apps. (b) Distribution of the number of "dangerous" permissions requested by malware, fraudulent and benign apps. (c) Dangerous permission ramp during version updates for a sample app "com.battery.plusfree". Originally the app requested no dangerous permissions.

which will easily be learned and detected by supervised learning algorithms.

IRR Features. We extract temporal features (see Table 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.

4.5 Jekyll-Hyde App Detection (JH) Module

Fig. 12a 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". Fig. 12b 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 percent of the legitimate apps request more than 5 dangerous permissions, while 16.5 percent of the malware apps and 17 percent of the fraudulent apps request more than 5 permissions. Perhaps surprisingly, most legitimate (69 percent), malware (76 percent) and fraudulent apps (61 percent) request between 1 and 5 dangerous permissions.

After a recent Google Play policy change [33], 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*. Fig. 12c 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 [34] 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) [35], Decision Trees (DT) (C4.5) and Random Forest (RF) [36], using 10-fold cross-validation [37]. 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.

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

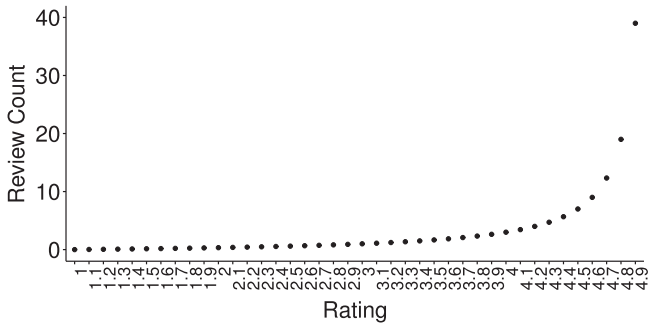

Fig. 13. Lower bound on the number of fake reviews that need to be posted by an adversary to cancel a 1-star review, versus the app's current rating (shown with 0.1-star granularity). At 4 stars, the adversary needs to post 3 5-star reviews to cancel a 1-star review, while at 4.2 stars, 4 5-star reviews are needed.

TABLE 3
Review Classification Results (10-Fold Cross-Validation) of Gold Standard Fraudulent (Positive) and Genuine (Negative) Reviews

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

MLP achieves the lowest false positive rate (FPR) of 1.47%.

TABLE 4
FairPlay Classification Results (10-Fold Cross Validation) of Gold Standard Fraudulent (Positive) and Benign Apps

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

RF has lowest FPR, thus desirable [38].

Section 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). 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 3 shows the results of the 10-fold cross validation of algorithms classifying reviews as genuine or fraudulent (Random Forest, Decision Tree and MLP). Fig. 14 shows the ROC plots of these algorithms. To minimize wrongful accusations, we seek to minimize the FPR [38]. MLP simultaneously achieves the highest accuracy of 96.26 percent and the lowest FPR of 1.47 percent (at 6.67 percent FNR). The EER of MLP is 3.6 percent 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.

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, 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 versus benign and malware versus benign apps.

We seek to identify the algorithms that achieve low FPR values, while having a reasonable FNR [38], [39]. 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 shows 10-fold cross validation results of FairPlay on the gold standard fraudulent and benign apps (see Section 3.2). All classifiers achieve an accuracy of around 97 percent. Random Forest is the best, having the highest accuracy of 97.74 percent and the lowest FPR of 1.01 percent. Its EER is 2.5 percent and the area under the ROC curve (AUC) is 0.993 (see Fig. 15).

Fig. 16a 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. [16]'s solution as a baseline to evaluate the ability of FairPlay to accurately detect malware. We computed Sarma et al. [16]'s RCP and RPCP indicators (see Section 2.1) using the longitudinal app dataset. We used the SVM based variant of Sarma et al. [16], which performs best. Table 4 shows 10-fold cross validation results over the malware and benign gold standard sets. FairPlay significantly outperforms Sarma

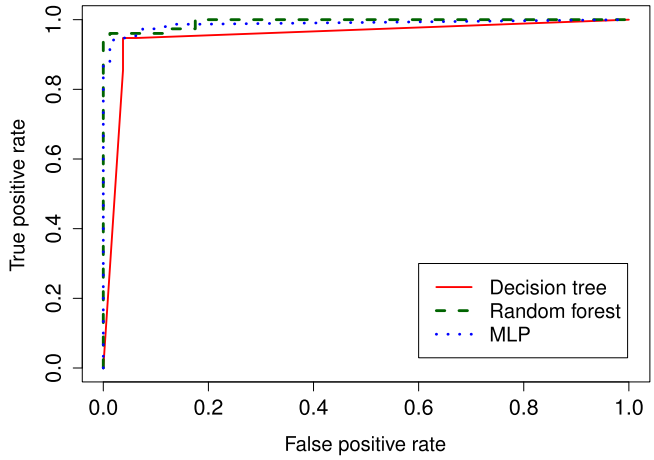

Fig. 14. ROC plot of 3 classifiers: Decision Tree, Random Forest, and Multilayer Perceptron (MLP). for review classification. RF and MLP are tied for best accuracy, of 96.26 percent. The EER of MLP is as low as 0.036.

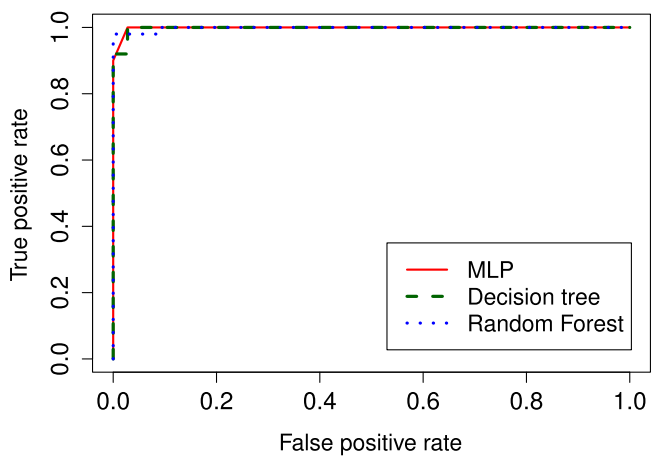

Fig. 15. ROC plot of 3 classifiers: Decision Tree, MLP, and Bagging for app classification (legitimate versus fraudulent). Decision Tree has the highest accuracy, of 98.99 percent. The EER of MLP is as low as 0.01.

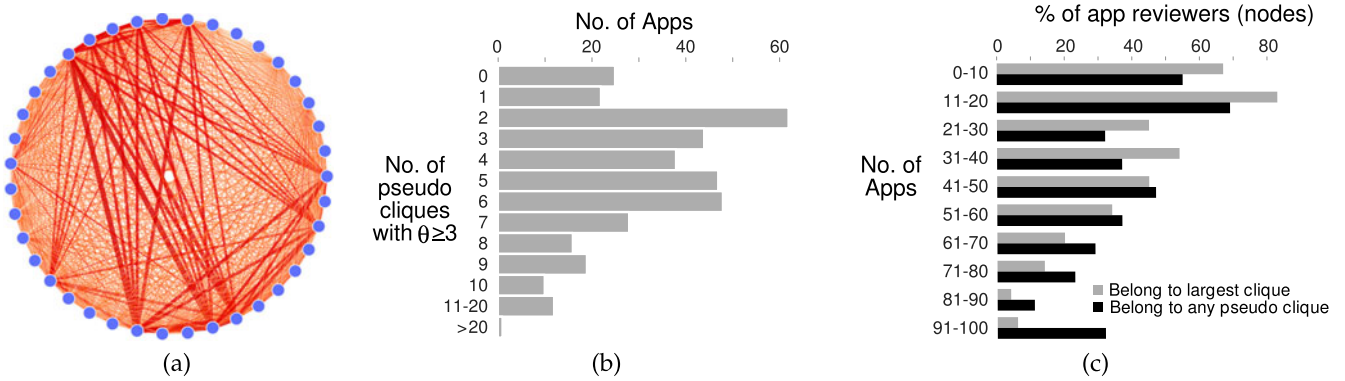

Fig. 16. (a) Clique flagged by PCF for "Tiempo - Clima gratis", one of the 201 seed fraud apps (see Section 3.2). The clique contains 37 accounts (names hidden for privacy) that reviewed the app. The edge weights are suspiciously high: any two of the 37 accounts reviewed at least 115 apps and up to 164 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 percent 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. Eight percent of the 372 apps have more than 90 percent of their reviewers involved in a clique!

et al. [16]'s solution, with an accuracy that consistently exceeds 95 percent. We note that the performance of Sarma et al.'s solution is lower than the one reported in [16]. This inconsistency may stem from the small number of malware apps that were used both in [16] (121 apps) and in this paper (212 apps).

For FairPlay, Random Forest has the smallest FPR of 1.51 percent and the highest accuracy of 96.11 percent. It also achieves an EER of 4 percent 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 percent of malware as fraud participants. Random Forest discovers 72.15 percent, and Decision Tree flags 75.94 percent 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 6 shows the most impactful features of FairPlay when using the Decision Tree algorithm to classify fraudulent versus benign and malware versus 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 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. Fig. 17 shows

TABLE 5
FairPlay Classification Results (10-Fold Cross Validation) of Gold Standard Malware (Positive) and Benign Apps, Significantly Outperform Sarma et al. [16]

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. [16]/SVM	65.32	24.47	55.23

FairPlay's RF achieves 96.11% accuracy at 1.51% FPR.

TABLE 6
Top Eight Most Important Features When Classifying Fraudulent versus Benign Apps (Center Column) and Malware versus Benign Apps (Rightmost Column)

Rank	Fraudulent versus Benign	Malware versus 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	$malW$	$dangerCount$

Notations are described in Table 1. While some features are common, some are more efficient in identifying fraudulent apps than malware apps, and vice versa.

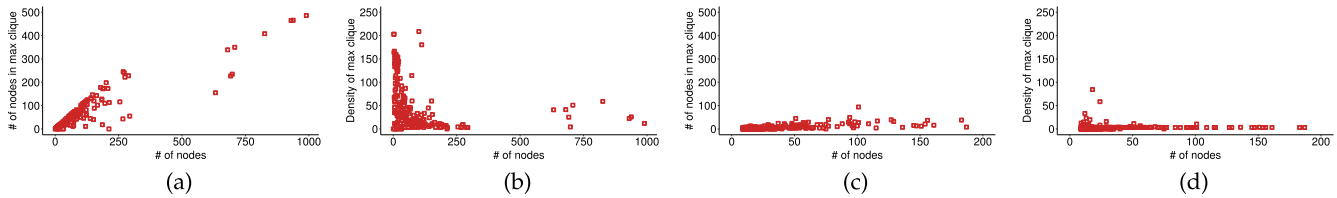

Fig. 17. Scatterplots for the gold standard fraudulent and malware apps. (a) Each red square represents a fraudulent app, whose y axis value is its number of nodes (reviews) in the largest pseudo-clique identified, and whose x axis value is its number of nodes. (b) For each fraudulent app, the density of its largest pseudo-clique versus its number of nodes. (c) For each malware app, the size of its largest pseudo-clique versus its number of nodes. (d) For each malware app, the density of its largest pseudo-clique versus 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.

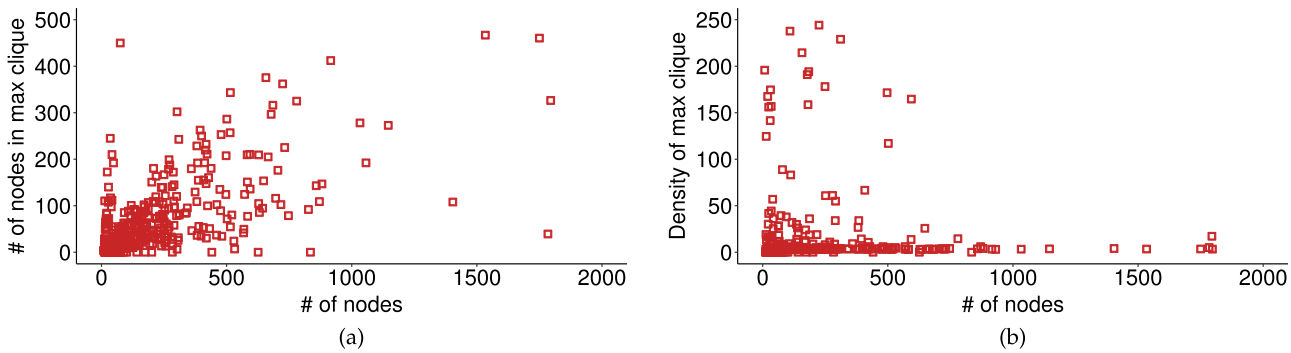

Fig. 18. Scatterplots of the 372 fraudulent apps out of 1,600 investigated, showing, for each app, (a) the number of nodes (reviews) in the largest clique identified versus the app's number of nodes and (b) the density of the largest clique versus the app's number of nodes. While apps with more nodes also tend to have larger cliques, those cliques tend to have lower densities.

scatterplots over the gold standard fraudulent and malware apps, of the sizes and densities of their largest pseudo-cliques, as detected by FairPlay. Fig. 17a shows that fraudulent apps tend to have very large pseudo-clique and Fig. 17c 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.

Fig. 17b shows that the largest pseudo-cliques of the larger fraudulent apps tend to have smaller densities. Fig. 17d 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.

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.

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 percent) as fraudulent. The Racing and Arcade categories have the highest fraud densities: 34 percent and 36 percent 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 percent of 1,600 apps). Fig. 16b shows that 93.3 percent of the 372 apps have at least 1 pseudo-clique of $\theta \geq 3$, nearly 71 percent have at least 3 pseudo-cliques, and a single app can have up to 23 pseudo-cliques. Fig. 16c shows that the pseudo-cliques are large and encompass many of the reviews of the apps: 55 percent of the 372 apps have at least 33 percent of their reviewers involved in a pseudo-clique, while nearly 51 percent of the apps have a single pseudo-clique containing 33 percent of their reviewers.

Fig. 18 shows the scatterplots of the number of nodes and densities of the largest clique in each of the 372 apps. While

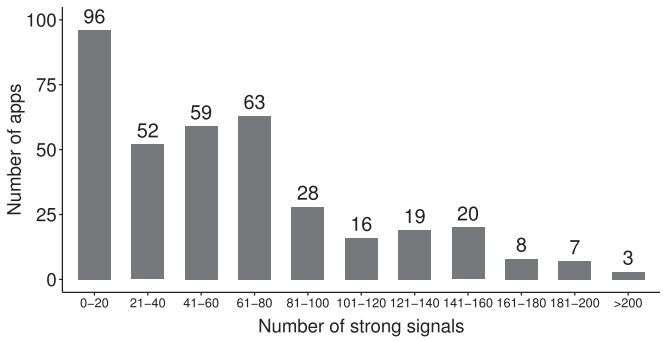

Fig. 19. Distribution of the number of malware and fraud indicator words (see Step RF.2) in the reviews of the 372 identified fraudulent apps (out of 1,600 apps). Around 75 percent of these apps have at least 20 fraud indicator words in their reviews.

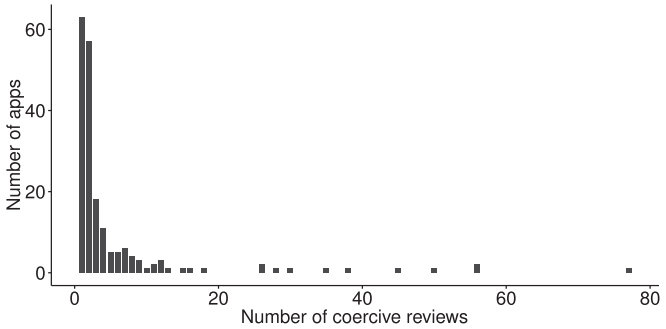

Fig. 20. Distribution of the number of coerced reviews received by the 193 coercive apps we uncovered. 5 apps have each received more than 40 reviews indicative of rating coercion, with one app having close to 80 such reviews!

intuitively apps with more reviews tend to have larger pseudo-cliques (Fig. 18a), surprisingly, the densities of such pseudo-cliques are small (Fig. 18b).

Fig. 19 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 percent of the 372 fraudulent apps have at least 20 fraud indicator words in their reviews.

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 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. Fig. 20 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.

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.

Acknowledgments

This research was supported in part by NSF grants 1527153, 1526254, and 1450619, and ARO W911NF-13-1-0142.

References

[1] Google Play. [Online]. Available: https://play.google.com/
[2] E. Siegel, "Fake reviews in Google Play and Apple App Store," Appentive, Seattle, WA, USA, 2014.
[3] Z. Miners. (2014, Feb. 19). "Report: Malware-infected Android apps spike in the Google Play store," *PC World*. Available: http://www.pcworld.com/article/2099421/report-malwareinfected-android-apps-spike-in-the-google-play-store.html
[4] S. Mlot. (2014, Apr. 8). "Top Android App a Scam, Pulled From Google Play," *PCMag*. Available: http://www.pcmag.com/article2/0,2817,2456165,00.asp
[5] D. Roberts. (2015, Jul. 8). "How to spot fake apps on the Google Play store," *Fortune*. Available: http://fortune.com/2015/07/08/google-play-fake-app/
[6] A. Greenberg (2012, May 23). "Researchers say they snuck malware app past Google's 'Bouncer' Android market scanner," Forbes Security, [Online]. Available: http://www.forbes.com/sites/andygreenberg/2012/05/23/researchers-say-they-snuck-malware-app-past-googles-bouncer-android-market-scanner/#52c8818d1041
[7] Freelancer. [Online]. Available: http://www.freelancer.com
[8] Fiverr. [Online]. Available: https://www.fiverr.com/
[9] BestAppPromotion. [Online]. Available: www.bestreviewapp.com/
[10] G. Wang, et al., "Serf and turf: Crowdturfing for fun and profit," in *Proc. ACM WWW*, 2012. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187928
[11] J. Oberheide and C. Miller, "Dissecting the Android Bouncer," presented at the *SummerCon2012*, New York, NY, USA, 2012.
[12] VirusTotal - free online virus, Malware and URL scanner. [Online]. Available: https://www.virustotal.com/, Last accessed on: May 2015.
[13] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani , "Crowdroid: Behavior-based Malware detection system for Android," in *Proc. ACM SPSM*, 2011, pp. 15–26.
[14] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for Android devices," *Intell. Inform. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.
[15] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proc. ACM MobiSys*, 2012, pp. 281–294.
[16] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android Permissions: A Perspective Combining Risks and Benefits," in *Proc. 17th ACM Symp. Access Control Models Technol.*, 2012, pp. 13–22.
[17] H. Peng, et al., "Using probabilistic generative models for ranking risks of Android Apps," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 241–252.
[18] S. Yerima, S. Sezer, and I. Muttik, "Android Malware detection using parallel machine learning classifiers," in *Proc. NGMAST*, Sep. 2014, pp. 37–42.
[19] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 95–109.
[20] Fraud detection in social networks, [Online]. Available: https://users.cs.fiu.edu/ carbunar/caspr.lab/socialfraud.html
[21] Google I/O 2013 - getting discovered on Google Play, 2013. [Online]. Available: www.youtube.com/watch?v=5Od2SuL2igA

[22] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," in *Proc. Eur. Intell. Secur. Inf. Conf.*, 2012, pp. 141–147.
[23] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *Proc. Int. Joint Conf. CISIS12-ICEUTE' 12-SOCO' Special Sessions*, 2013, pp. 289–298.
[24] J. Ye and L. Akoglu, "Discovering opinion spammer groups by network footprints," in *Machine Learning and Knowledge Discovery in Databases*. Berlin, Germany: Springer, 2015, pp. 267–282.
[25] L. Akoglu, R. Chandy, and C. Faloutsos, "Opinion Fraud Detection in Online Reviews by Network Effects," in *Proc. 7th Int. AAAI Conf. Weblogs Soc. Media*, 2013, pp. 2–11.
[26] Android market API, 2011. [Online]. Available: https://code.google.com/p/android-market-api/
[27] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theory. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2006.06.015
[28] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in *Proc. 9th Scandinavian Workshop Algorithm*, 2004, pp. 260–272.
[29] T. Uno, "An efficient algorithm for enumerating pseudo cliques," in *Proc. ISAAC*, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1781574.1781621
[30] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. Sebastopol, CA, USA: O'Reilly, 2009.
[31] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs Up? sentiment classification using machine learning techniques," in *Proc. ACL-02 Conf. Empirical Methods Natural Lang. Process.*, 2002, pp. 76–86.
[32] J. H. McDonald, *Handbook of Biological Statistics*, 2nd ed. Baltimore, MD, USA: Sparky House Publishing, 2009. [Online]. Available: http://udel.edu/∼mcdonald/statintro.html
[33] New Google Play Store greatly simplifies permissions, 2014. [Online]. Available: http://www.androidcentral.com/new-google-play-store-4820-greatly-simpli
[34] Weka. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/
[35] S. I. Gallant, "Perceptron-based learning algorithms," *Trans. Neur. Netw.*, vol. 1, no. 2, pp. 179–191, Jun. 1990.
[36] L. Breiman, "Random Forests," *Mach. Learning*, vol. 45, pp. 5–32, 2001.
[37] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, 1995, pp. 1137–1143.
[38] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining and inference for malware detection," in *Proc. SIAM Int. Conf. Data Mining*, 2011, Art. no. 12.
[39] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by association: Large scale malware detection by mining file-relation graphs," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1524–1533. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623342

Mahmudur Rahman received the PhD degree in computer science from Florida International University. He is now a security architect in the Watson Security team at IBM. His current work focuses on the design of business-driven cognitive solutions that meet security requirements related to function, protection, assurance, risk management, and compliance. His research interests include security and privacy with applications in online and geosocial networks, wireless networks, distributed computing systems, and mobile applications.

Mizanur Rahman is working toward the PhD degree at Florida International University. He has previously held various positions at KAZ Software, iAppDragon, and Prolog, Inc. His research interest include internet data privacy, fraud detection in social network, and user experience analysis.

Bogdan Carbunar received the PhD degree in computer science from Purdue University. He is an assistant professor in SCIS at FIU. Previously, he held various researcher positions within the Applied Research Center at Motorola. His research interests include distributed systems, security, and applied cryptography.

Duen Horng Chau received the master's degree in human-computer interaction and the PhD degree in machine learning. He is an assistant professor at Georgia Tech's School of Computational Science and Engineering, and an associate director of the MS Analytics program. His PhD thesis won Carnegie Mellon's Computer Science Dissertation Award, Honorable Mention. He received faculty awards from Google, Yahoo, and LexisNexis. He also received the Raytheon Faculty Fellowship, Edenfield Faculty Fellowship, and the Outstanding Junior Faculty Award. He is the only two-time Symantec fellow and an award-winning designer.