Dissecting the Bouncer and Malware Detection in Google Play

[1] A.Kiranmayi, [2]N.Padmaja.
[1]PG Student, Dept. of CSE, School of Engineering & Technology, Sri Padmavati Mahila University (Women's University), Tirupati .
[2]Assistant Prof, M.Tech, Dept. of CSE, Sri Padmavati Mahila University (Women's University), Tirupati .

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 (87K apps, 2.9M reviews, and 2.4M reviewers, collected over half a year), in order to identify suspicious apps. 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. 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

I 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]–[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 Figure 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% (948) were flagged by at least one anti-virus tool and 2% (150) were identified as malware by at least 10 tools (see Figure 6). Previous mobile malware detection work has focused on dynamic analysis of app executables [13]–[15] as well as static analysis of code and permissions [16]–[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 combinationFig. 1: An "install job" posting from Freelancer [7], asking for 2000 installs within 3 days (in orange), in an organized way that includes expertise verifications and provides secrecy assurances (in blue). Text enlarged for easier reading. 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.

We propose FairPlay, a system that leverages the above observations to efficiently detect Google Play fraud and malware (see Figure 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 [§ 4]:

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

FairPlay also enabled us to discover a novel, coercive review campaign attack type, where app users are harassed into writing a positive review for the app, and install and review other apps. We have discovered 1, 024 coerced reviews, from users complaining about 193 such apps [§ 5.4 & § 5.5].

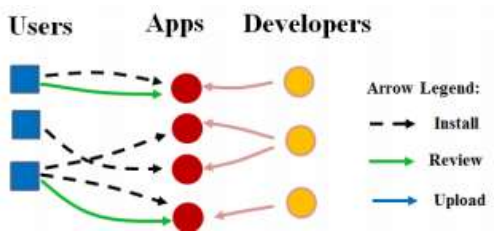

Fig. 1: Google Play components and relations. Google Play's functionality centers on apps, shown as red disks. Developers, shown as orange disks upload apps. A developer may upload multiple apps. Users, shown as blue squares, can install and review apps. A user can only review an app that he previously installed.

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]–[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 vs. benefits tradeoffs of apps. In § 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 classi- fier on 2000 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.

II 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.

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.

A. 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 addon 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.

III Longitudinal App Data

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

By monitoring new apps, we aim to capture in real-time the moments when such search rank fraud campaigns begin.

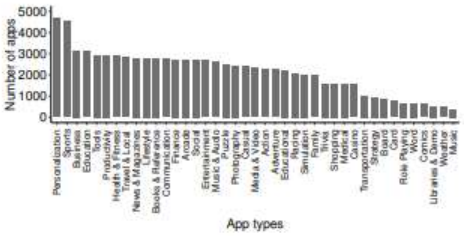

Fig. 3: Distribution of app types for the 87, 223 fresh app set. With the slight exception of "Personalization" and "Sports" type spikes, we have achieved an almost uniform distribution across all app types, as desirable.

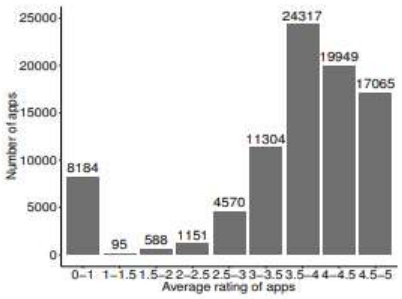

Fig. 4: Average rating distribution for the 87, 223 fresh app set. Most apps have more than 3.5 stars, few have between 1 and 2.5 stars, but more than 8, 000 apps have less than 1.

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

Figure 3 shows the distribution of the fresh app categories. Figure 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 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.

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 ap

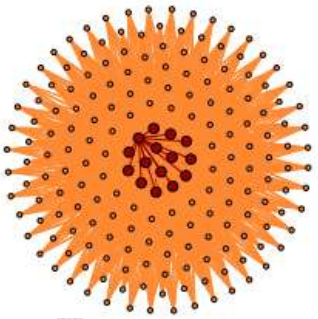

Fig. 5: Co-review graph of 15 seed fraud accounts (red nodes) and the 188 GbA accounts (orange nodes).

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.

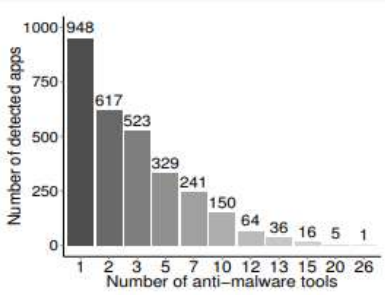

Fig. 6: Apks detected as suspicious (y axis) by multiple anti-virus tools (x axis), through VirusTotal [12], from a set of 7, 756 downloaded apks.

proach enables us to overcome the Google Play limit of 4000 displayed reviews per app: each snapshot will capture only the reviews posted after the previous snapshot.

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 setof 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.

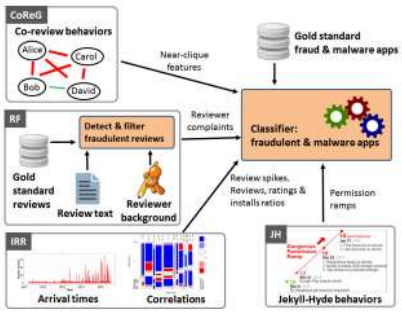

Fig. 7: FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies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

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).

FairPlay organizes the analysis of longitudinal app data into the following 4 modules, illustrated in Figure 7. The CoReview 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 summarizes the most important features. We now detail each module and the features it extracts.

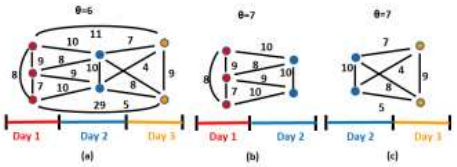

Fig. 8: Example pseudo-cliques and PCF output. Nodes are users and edge weights denote the number of apps reviewed in common by the end users. Review timestamps have a 1-day granularity. (a) The entire co-review graph, detected as pseudo-clique by PCF when θ is 6. When θ is 7, PCF detects the subgraphs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

IV The Co-Review Graph (CoReG) Module

This module exploits the observation that fraudsters who control many accounts will re-use them across multiple jobs. Its goal is then to detect sub-sets of an app's reviewers that have performed significant common review activities in the past. In the following, we describe the co-review graph concept, formally present the weighted maximal clique enumeration problem, then introduce an efficient heuristic that leverages natural limitations in the behaviors of fraudsters. Co-review graphs. Let the co-review graph of an app, see Figure 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. Figure 16a shows the co-review clique of one of the seed fraud apps (see § 3.2). The co-review graph concept naturally identifies user accounts with significant past review activities.

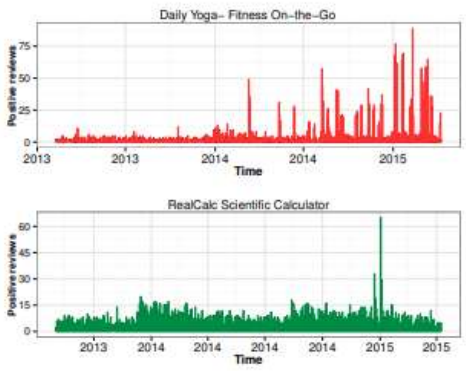

Fig. 9: Timelines of positive reviews for 2 apps from the fraudulent app dataset. The first app has multiple spikes while the second one has only one significant spike. sentences extracted from randomly selected 350 positive and 410 negative Google Play reviews, and (ii) 10, 663 sentences extracted from 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% and a false positive rate of 19.65%, for an overall accuracy of 81.74%. 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% confidence interval for our result. The deviation of the binomial distribution is 0.011. Thus, we are 95% confident that the true performance of the classifier is in the interval (79.55, 83.85).

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

In Section V 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 III.

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 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".

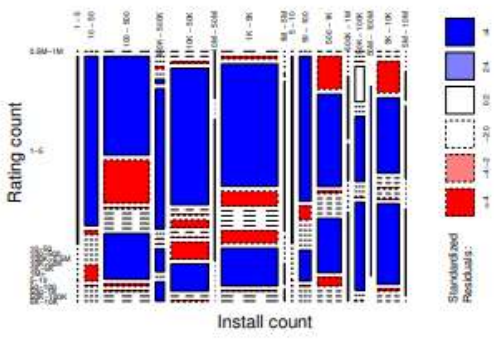

Fig. 10: Mosaic plot of install vs. rating count relations of the 87K apps. Larger cells (rectangles) signify that more apps have the corresponding rating and install count range; dotted lines mean no apps in a certain install/rating category. The standardized residuals identify the cells that contribute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

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.

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,

Claim 1. Let RA 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 RA−1 5−RA positive reviews.

Proof: Let σ be the sum of all the k reviews received by a before time T . Then, RA = σ k . Let qr be the number of fraudulent reviews received by A. To compensate for the 1 star review posted at time T , qr is minimized when all those reviews are 5 star. We then have that: RA = σ k = σ+1+5qr k+1+qr . 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, weobtain that qr = σ−k 5k−σ = RA−1 5−RA . The last equality follows by dividing both the numerator and denominator by k.

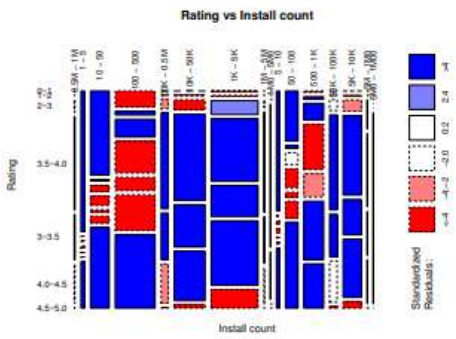

Fig. 11: Mosaic plot showing relationships between the install count and the average app rating, over the 87K apps. A cell contains the apps that have a certain install count interval (x axis) and rating interval (y axis). Larger cells contain more apps. We observe a relationship between install count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

The 1, 024 coerced reviews were posted for 193 apps. Figure 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.

V 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.

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] Freelancer. http://www.freelancer.com.
[8] Fiverr. https://www.fiverr.com/.
[9] BestAppPromotion. www.bestreviewapp.com/.
[10] Gang Wang, Christo Wilson, Xiaohan Zhao, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y. Zhao. Serf and Turf: Crowdturfing for Fun and Profit. In *Proceedings of ACM WWW*. ACM, 2012.
[11] Jon Oberheide and Charlie Miller. Dissecting the Android Bouncer. *SummerCon2012, New York*, 2012.
[12] VirusTotal - Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/, Last accessed on May 2015.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] Fraud Detection in Social Networks. https://users.cs.fiu.edu/ _carbunar/caspr.lab/socialfraud.html.

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

[22] Justin Sahs and Latifur Khan. A Machine Learning Approach to Android Malware Detection. In *Proceedings of EISIC*, 2012.

[23] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo ´Alvarez. Puma: Permission usage to detectmalware in android. In *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pages 289–298. Springer, 2013.

[24] 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.

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

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

[27] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worstcase time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, October 2006.

[28] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. 3111:260–272, 2004.

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

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

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

[32] John H. McDonald. *Handbook of Biological Statistics*. Sparky House Publishing, second edition, 2009.

A.Kiranmayi was born in AP, India. Currently she is studying her Post graduate degree in School of Engineering & Technology, Sri Padamavathi Mahila visvavidyalayam, Tirupathi in Department of Computer Science & Engineering.

N.Padmaja is currently working as an Assistant Professor in CSE department, School of Engineering & Technology, Sri Padamavathi Mahila visvavidyalayam,Tirupati.