**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**
**IMPACT FACTOR: 6.017**

***IJCSMC, Vol. 7, Issue. 6, June 2018, pg.154 – 164***

# SECURING GOOGLE PLAY STORE FROM FAKE REVIEWS AND MALWARE ATTACK BY ANALYZING THE USER BEHAVIORS

**Mr. Ali Askar K.M, Mr. Unnikrishnan S Kumar**
Dept. of Computer Science and Engineering, APJ Abdul Kalam Technological University, India
Dept. of Computer Science and Engineering, APJ Abdul Kalam Technological University, India
Aliaskarkm786@gmail.com; uksknair@gmail.com

***Abstract— In this digital age, smart phones are become one of the important equipment for each and every people irrespective of their financial or social status. By increasing the popularity of the smart phones, the applications that runs on smart phones are also becomes popular and highly demanding. Google Play, the most popular Android app market which working with 87 K apps, 2.9 M reviews, and 2.4M reviewers( collected over half a year). By the popularity and acceptance among the users, Google play is becomes one of the important financial hub in App marketing and also a good source of Virus and Malware spreading over the world. To face these problems, we propose a simple but largely affective approach that "Securing Google Play Store from Fraud and Malware Attack by Analyzing the User Behaviors". By using this method, we find out that we are track down hundreds of fraudulent apps and fake reviewers that are currently evade by Google bouncer's technology.***

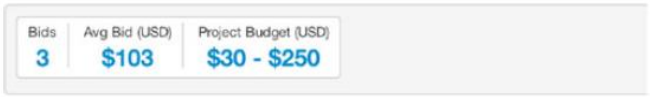
***Keywords- component, Android app Market, Google play, Malware, Google bouncer***

## I. INTRODUCTION

By the increasing demand and usage of Android app markets such as Google Play [1] and the incentive model they offer to popular apps, make them effective and interesting targets for fraudulent and malicious behaviors'. Some fake developers are sneaky boost the search rank and popularity of their apps (e.g., through fake reviews and bogus installation counts) [2], while malware developers use app markets as a spreading safe sector for their malware [3]. The motivation for such behaviors' is impact: app popularity and demand translate into financial benefits and expedited malware propagation. sneaky developers frequently exploit crowd sourcing sites (e.g., Freelancer [4], Fiverr [5] ) to hire teams of willing workers to make fraud collectively, doing realistic, spontaneous activities from unrelated people (i.e., "crowd turfing" [6]), see Fig. 1 for an example. We call this behavior "search rank fraud". By considering the efforts of Android markets to identify and remove malware are not always successful. Google Play uses the Bouncer system [7] to remove malware. However, out of the 7,756 Google Play

apps we analysed using VirusTotal[8] , 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. 2). Currently available mobile malware detection work has mainly concentrating on dynamic analysis of app execution [9] as well as static analysis of code and permissions [10]. But, recent Android malware analysis revealed that malware generates quickly to bypass anti-virus tools [11]. In this paper, we seek to identify both malware and search rank fraud subjects in Google Play. We concentrate on malicious developers who make fraud attempt as search rank fraud to boost the impact of their malware. Unlike existing solutions, we build this work on the analysis that fraud and malicious behaviors' are make highly negative impact on app markets. We disclose these unauthorized acts by picking out such attempts. Considering the high expense of setting up authorizedGoogle 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. Real users affected by malware may report unpleasant experiences in their reviews.




**Project Description:**

We are a Mobile Game Development firm, looking for promotion of our Mobile game across Google Play Store. We are looking for someone who can get us upto 2000 installs within the first 3 days of the release.

If you can provide this service in different quantities, mention it along with your bid or contact me directly with your offer.

Bidders are advised to be ready with examples of their previous projects of such nature or verify the authenticity of their methods.

Details of the projects may be shared with prospective candidates, but the actual product URLs with only be shared with the selected candidates.

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.

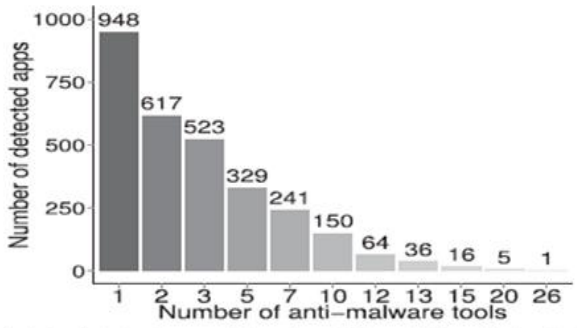



Fig. 2. Apks detected as suspicious ($y$ axis) by multiple anti-virus tools ($x$ axis), through VirusTotal [12], from a set of 7,756 downloaded apks.

## II. LITERATURE SURVEY

For this study we are focused on the Google play store which is the Android app. 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, install count range, 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. 3 represents the participants in Google Play and their relations.

. We consider not only malicious developers, who upload malware, but also rational fraudulent developers. Fraud developers attempt to intrude with the search rank of their apps, e.g., by making fraud experts in crowd sourcing sites to write fake 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. 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 frauds, 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.

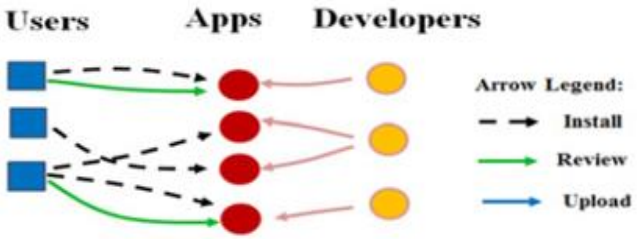



Fig. 3 . Google Play components and relations. Google Play's functionality centers on apps, shown as red disks. Developers, shown as orange disks upload apps. A developer may upload multiple apps. Users, shown as blue squares, can install and review apps. A user can only review an app that he previously installed.

## III.METHODOLOGY

We have collected data from 10K+ newly released apps over more than 2 months, and identified 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 not allows scripts from scrolling down a user page. Thus, to collect the id's of more than 20 apps reviewed by a user. To overcome this issue, we developed a php 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 communicates with Google Play pages using content scripts 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 take 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 takes the list of apps rated or reviewed by the user.

We have also created the Application downloader (AD), a Java tool to automatically download applications offree apps on a PC, using the open-source Android Market API [26]. AD takes as input a list of free app ids, a Gmail account and password, and a GSF id. AD makes a new market place for the "androidprotect" service and logs in AD sets contents for the session context, then issues a GetAssetRequest for each app checker in the input list. AD introduces a 15s waiting between each requests. The result contains the url for the app; AD uses this url to take and keep the app's binary stream into a local file. After taking the binaries of the apps on the list, AD scans each app application files using VirusTotal [12] (which is an online malware detector provider, to find out the number of anti-malware tools) that identify the application files as suspicious.

**App Data Collection**: To find skeptical changes that occur early in the endurance of apps, we used the "New Releases" link to identify apps with a short history on Google Play. Our concern in newly released apps trunk from our analysis of search rank fraud jobs posted on crowd sourcing sites, that revealed that app developers often recruit frauds early after uploading their apps on Google Play. Their motive is likely to create the illusion of an up-and-coming app, that may then snowball with interest from real users. By analyzing new apps, we aim to capture in real-time the moments when such search rank fraud attempts begin.

We assumes the first upload date of an app using the day of its first review. We have started taking new releases in July 2017 and by October 2017 we had a set of 10,234 apps, whose first upload time was under 30 days prior to our first collection time, when they had at most 80 reviews. 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). Most apps have at least a 3.5 rating aggregate rating, with few apps between 1 and 2.5 stars. However, we observe that more than 3,000 apps with less than 1 star.

**Good Standard Data:**

**Malware Apps:** We used AD (see Section III) to collect the apks of 1,756 randomly selected apps from the set . 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" data set, for a total of 212 apps.

**Fraud Apps:** We take the contacts established among search rank fraud community, to identify the identities of 10 Google Play accounts that were used to write fraud reviews for 201 unique apps. We call the 10 accounts "fraud accounts" and the 201 apps "fraud apps". Fig. 4 shows the graph formed by the review habits of the 10 accounts: nodes are accounts, edges connect accounts who reviewed apps in common, and edge weights represent the number of such commonly reviewed apps. The 10 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 fraud accounts range between 60 and 217.

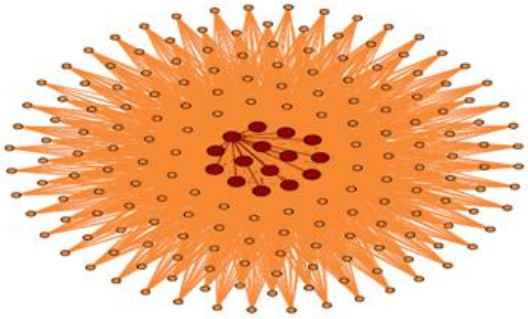


Fig. 4 . Co-review graph of 15 seed fraud accounts (red nodes) and the 188 GbA accounts (orange nodes). Edges indicate reviews written in common by the accounts corresponding to the endpoints. We only show edges having at least one seed fraud account as an endpoint. The 15 seed fraud accounts form a suspicious clique with edges weights that range between 60 and 217. The GbA accounts are also suspiciously well connected to the seed fraud accounts: the weights of their edges to the seed fraud accounts ranges between 30 and 302.

**Fraud Reviews:** We have taken all the 34098 reviews received by the 125 fraud apps. The 10 fraud accounts were responsible for 789 of these reviews. We used the 34098 reviews to identify 88 accounts, such that each account was used to review at least 10 of the 201 fraud apps We call these, guilt by association (GbA) accounts. Fig. 4 shows the co-review edges between these GbA accounts (in orange) and the fraud accounts: the GbA accounts are suspiciously well connected to the seed fraud accounts, with the weights of their edges to the accounts ranging between 12 and 190. To avoid feature ambiguty, we have used the 789 fraud reviews written by the 10 accounts and the 3488 fraud reviews written by the 88 GbA accounts for the 201 seed fraud apps, to extract a balanced set of fraudulent 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 1).

## IV. MODELLING

**User Analysis: Proposed Solution:** We now introduce User Analysis, a system to detect malicious and fraud apps. Overview of User Analysis organizes the analysis of app data into the following 4 modules, illustrated in Fig. 5. 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. User Analysis 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.

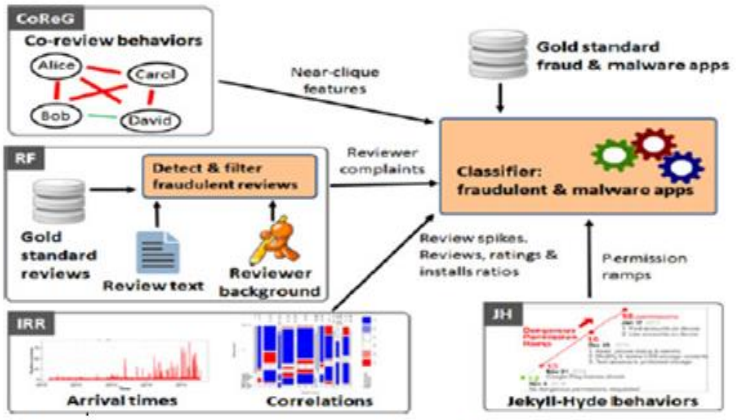



Fig. 5 . FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.

**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, 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. 6, 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. The coreview graph concept naturally identifies user accounts with significant past review activities.

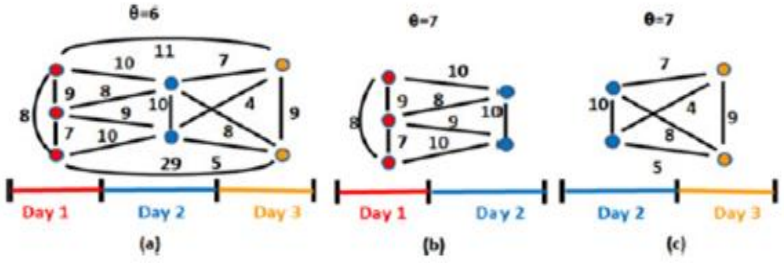



Fig.6 . Example pseudo-cliques and PCF output. Nodes are users and edge weights denote the number of apps reviewed in common by the end users. Review timestamps have a 1-day granularity.

**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 -> R that assigns a weight to each edge of G. Given a vertex sub-set U €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 cliqueif 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.

**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 u. PCF outputs a set of identified pseudo-cliques with r _ u, that were formed during contiguous time frames. In Section 5.3 we discuss the choice of u. 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 "working- 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 u (lines 6 and 23 _ 27). When no new nodes have been added to the work-in-progress pseudo-clique (line 8), we add the pseudo-clique to the output (line 9), then move to the next day (line 1). The greedy choice (getMaxDensityGain, not depicted in Algorithm 1) picks the review not yet in

the work-in-progress pseudo clique, whose writer has written the most apps in common with reviewers already in the pseudo-clique.

**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
```

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

| Notation | Definition |
|---|---|
| **CoReG Module** | |
| $nCliques$ | number of pseudo-cliques with $\rho \geq \theta$ |
| $\rho_{max}$, $\rho_{med}$, $\rho_{SD}$ | 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 |

**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.

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

| Notation | Definition |
|---|---|
| $\rho_R$ | 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 $\rho_R$ 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+ |

**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.

**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 behaviours.

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

**Jekyll-Hyde App Detection (JH) Module :**. 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. Perhapssurprisingly, 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 [12], 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.

## V. CONCLUSIONS

We have introduced User Analysis, 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 User Analysis. In addition, we showed User Analysis'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. [Online]. Available: https://play.google.com/
[2] E. Siegel, "Fake reviews in Google Play and Apple App Store," Appentive, Seattle, WA, USA, 2014.
[3] 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-snuckmalware-app-past-googles-bouncer-android-market-scanner/#52c8818d1041 [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
[4] Freelancer. [Online]. Available: http://www.freelancer.com
[5] Fiverr. [Online]. Available: https://www.fiverr.com/
[6] 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
[7] J. Oberheide and C. Miller, "Dissecting the Android Bouncer," presented at the SummerCon2012, New York, NY, USA, 2012.
[8] VirusTotal - free online virus, Malware and URL scanner.
[Online]. Available: https://www.virustotal.com/, Last accessed on: May 2015.
[9] 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.
[10] 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.
[11] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in Proc. IEEE Symp. Secur. Privacy, 2012, pp. 95–109.
[12] New Google Play Store greatly simplifies permissions, 2014. [Online].Available:http://www.androidcentral.com/newgoogle-play-store-4820-greatly-simpli