MALWARE AND FAKE APPLICATIONS IDENTIFY IN GOOGLE PLAY

[1]H ATEEQ AHMED., M. Tech ,Assistant Professor
[2]SHAIK ABDUL JABBAR., M. Tech, Assistant Professor
[3]A.EMMANUEL RAJU., M. Tech, Assistant Professor
DEPARTMENT OF CSE
DR K V SUBBA REDDY INSTITUTE OF TECHNOLOGY, DUPADU, KURNOOL

ABSTRACT - The use of mobile devices including Tablets, Smart watch, and note books are increasing day by day. Android has the major share in the mobile application market. Android mobile applications become an easy target for the attackers because of its open source environment. Also user's ignorance the process of installing and usage of the apps. To identify fake and malware applications, all the previous methods focused on getting permission from the user and executing that particular mobile application. A malware detection framework that discovers and break traces left behind by fraudulent developers, to detect search rank fraud as well as malware in Google Play. The fraud app is detected by aggregating the three pieces of evidence such as ranking based, co-review based and rating based evidence. Finally aggregating all the activities of front running apps, it can be achieve certain accuracy in classifying benign standard datasets of malware, fraudulent and legitimate apps. Additionally, we apply incremental learning approach to characterize a large number of data sets. It combined effectively for all the evidences for fraud detection. To accurately locate the ranking fraud, there is a need to mining the active period's namely leading sessions, of mobile Apps.

I. INTRODUCTION

In a recent trend, instead of relying on traditional marketing solutions, shady App developers resort to some fraudulent means to deliberately boost their Apps and eventually manipulate the chart rankings on an App store. This is usually implemented by using so-called ―bot farms‖ or ―human water armies‖ to inflate the App downloads, ratings and reviews in a very short time. Indeed, our careful observation reveals that mobile Apps are not always ranked high in the leader board, but only in some leading events, which form different leading sessions. Note that we will introduce both leading events and leading sessions in detail later. In other words, ranking fraud usually happens in these leading sessions. Therefore, detecting ranking fraud of mobile Apps is actually to detect ranking fraud within leading sessions of mobile Apps. automatically detect ranking fraud without using any benchmark information. Finally, due to the dynamic nature of chart rankings, it is not easy to identify and confirm the evidences linked to ranking fraud, which motivates us to discover some implicit fraud patterns of mobile Apps as evidences. Indeed, our careful observation reveals that mobile Apps are not always ranked high in the leader board, but only in some leading events, which form different leading sessions. Note that we will introduce both leading even and leading sessions in detail later. In other words, ranking fraud usually happens in these leading sessions.

Therefore, detecting ranking fraud of mobile Apps is actually to detect ranking fraud within leading sessions of mobile Apps. Specifically, we first propose a simple yet effective algorithm to identify the leading sessions of each App based on its historical ranking records. Then, with the analysis of Apps' ranking behaviors, we find that the fraudulent Apps often have different ranking patterns in each leading session compared with normal Apps. Thus, we characterize some fraud evidences from Apps' historical ranking records, and develop three functions to extract such ranking based fraud evidences. Nonetheless, the ranking based evidences can be affected by App developers' reputation and some legitimate marketing campaigns, such as ―limited-time discount‖. As a result, it is not sufficient to only use ranking based evidences. The number of mobile Apps has grown at a breathtaking rate over the past few years. For example, as of the end of April 2013, there are more than 1.6 million Apps at Apple's App store and Google Play. To stimulate the development of mobile Apps, many App stores launched daily App leader boards, which demonstrate the chart rankings of most popular Apps. Indeed, the App leader board is one of the most important ways for promoting mobile Apps. A higher rank on the leader board usually leads to a huge number of downloads and million dollars in revenue. Therefore, App developers tend to explore various ways such as advertising campaigns to promote their Apps in order to have their Apps ranked as high as possible in such App leader boards.

However, as a recent trend, instead of relying on traditional marketing solutions, shady App developers resort to some fraudulent means to deliberately boost their Apps and eventually manipulate the chart rankings on an App store. This is usually implemented by using So called ― bot farms ‖ or ― human water armies‖ to inflate the App downloads, ratings and reviews in a very short time. For example, an article from Venture Beat reported that, when an App was promoted with the help of ranking manipulation, it could be propelled from number 1,800 to the top 25 in Apple's top free leader board and more than 50,000-100,000 new users could be acquired within a couple of days.In fact, such ranking fraud raises great concerns to the mobile App industry. For example, Apple has warned of cracking down on App developers who commit ranking fraud in the Apple's App store. In the literature, while there are some related works, such as web ranking spam detection online review spam detection and mobile App recommendation, the problem of detecting ranking fraud for mobile Apps is still under explored. To fill this crucial void, in this paper, we propose to develop a ranking fraud detection system for mobile Apps. Along this line, we identify several

important challenges. First, ranking fraud does not always happen in the whole life cycle of an App, so we need to detect the time when fraud happens. Such challenge can be regarded as detecting the local anomaly instead of global anomaly of mobile Apps.

II. LITERATUR REVIEW

In paper [3] the author proposes the static method to detect the malware in mobile applications. In this system using reverse engineering concept the source code for the suspicious APK files. After that using structured mapping author builds the structure for the classes. Finally using data flow concept several patterns for the different type of threats has been created and use them to detect the malware in applications. Depending upon the number of threading pattern the effectiveness of this method is calculated.

In paper [1] the author proposed a new method to detect malware in mobile applications by examining the runtime behavior of that particular application in the mobile environment. The author proposes that unexpected behavior mobile app can vary from one application to other applications. Also, it varies from the environment of that particular application running on different devices. Using Xposed framework user can change the user and system application without modifying the application package(APK).Depend upon that user can set particular conditions to identify the malware in the mobile applications.

In paper [2] the author proposes some of modern machine learning algorithms to detect malware. For that these algorithms are applied to the metadata collected from the Google Play. While all of the existing methods for detecting algorithm focused on inherent characteristics of the particular mobile app this gives a direct method to detect the applications. For the setup of the experiments the collected 25k data from Google Play. Developers update their applications in particular interval of days whereas fake applications could not be updated since its upload of the Google Play. All of these works focused on only linear models Future work may focus on non-liners models.

In paper [5] the author aims to protect the review spanners or spam reviews. The spammer may target only on the specific protect. After that, they gave fake reviews to that particular mobile app by creating the different account to review that account. The author proposes a novel based scoring method to detect every single review of the particular product. The author creates highly suspicious as a subset. By using web-based spammer evaluation software the fakeness of the review is calculated. After the completion of the evaluation, the result shows the effective to predict the fake reviews.

III. SYSTEM DESIGN

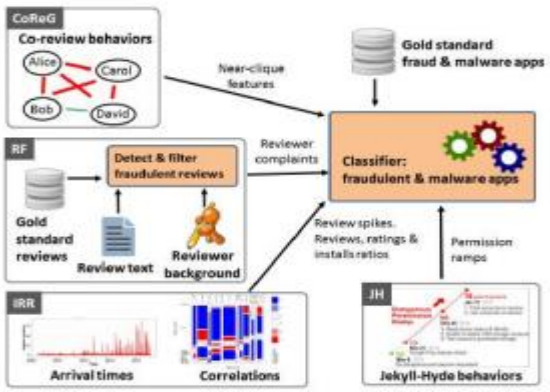

Fig1: Architecture Diagram

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 such as the app's average rating, total number of reviews, ratings and installs, for a total of 28 features. The Co-ReG 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 permissions ramps to pinpoint possible JekyllHyde app transitions. There are 200 apps that have more than 10 reviews and were developed by reputable media outlets. We have also collected the 32,022 reviews of these apps.

IV. 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.

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.

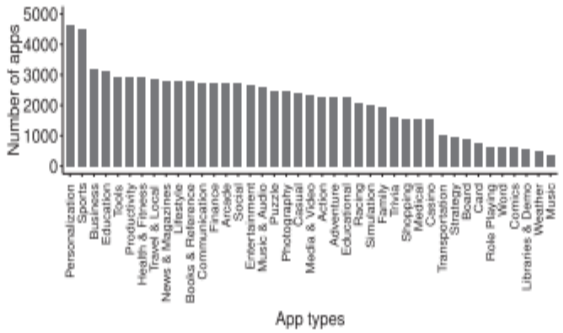

Fig2: Distribution of app types for the 87,223 fresh app set. With the slight exception of "Personalization" and "Sports" type spikes, we have achieved an almost uniform distribution across all app types, as desirable.

Above Fig2. 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

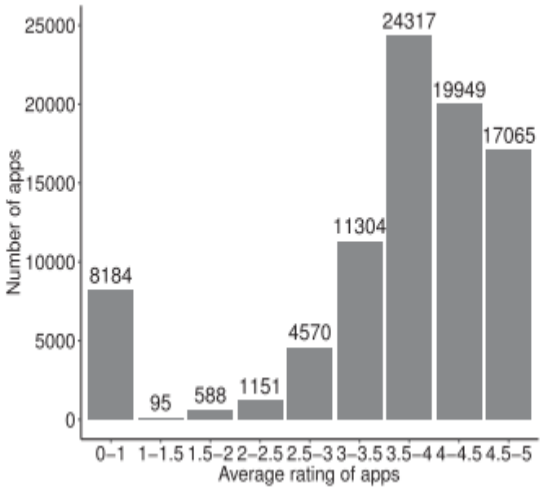

Fig3: Average rating distribution for the 87,223 fresh app set. Most apps have more than 3.5 stars, few have between 1 and 2.5 stars, but more than 8,000 apps have less than 1.

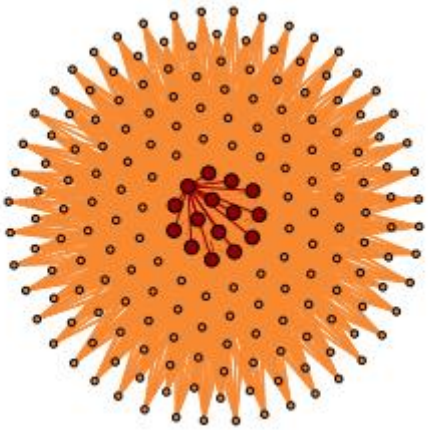

Fig4: Co-review graph of 15 seed fraud accounts (red nodes) and the 188 GbA accounts (orange nodes). Edges indicate reviews written in common by the accounts corresponding to the endpoints. We only show edges having at least one seed fraud account as an endpoint. The 15 seed fraud accounts form a suspicious 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.

V. FAIRPLAY: PROPOSED SOLUTION

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

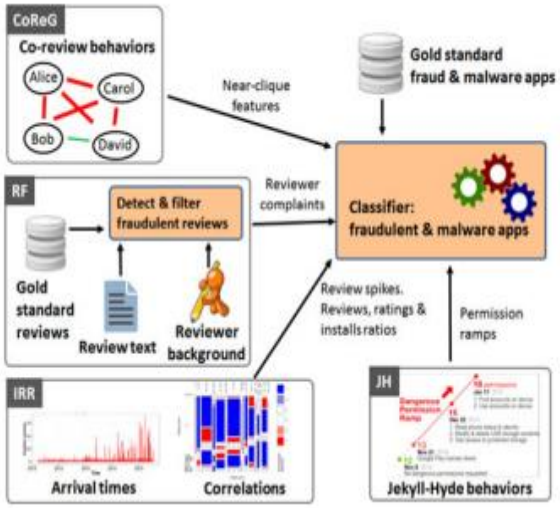

Fig5: FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible JekyllHyde app transitions

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

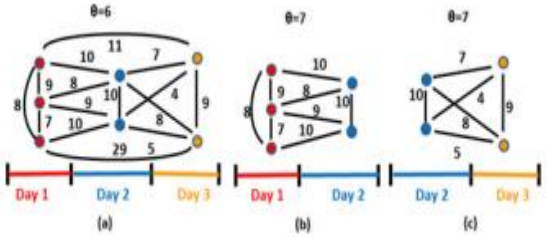

Fig6: Example pseudo-cliques and PCF output. Nodes are users and edge weights denote the number of apps reviewed in common by the end users. Review timestamps have a 1-day granularity. (a) The entire co-review graph, detected as pseudo-clique by PCF when u is 6. When u is 7, PCF detects the subgraphs of (b) the first two days and (c) the last two days. When u=8, PCF detects only the clique formed by the first day reviews (the red nodes).

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. 6 shows the co-review clique of one of the seed fraud apps. The coreview graph concept naturally identifies user accounts with significant past review activities. The Weighted Maximal Clique Enumeration Problem. Let G ¼ ðV; EÞ 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 2 V , we use G½U 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.

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
```

VI. EVALUATION

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 u 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 crossvalidation [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

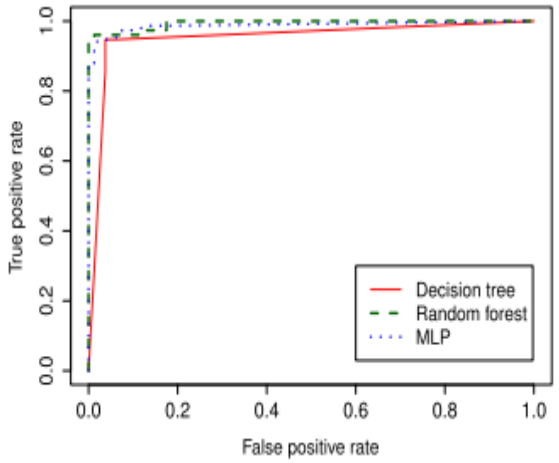

Fig7: ROC plot of 3 classifiers: Decision Tree, Random Forest, and Multilayer Perceptron (MLP). for review classification. RF and MLP are tied for best accuracy, of 96.26 percent. The EER of MLP is as low as 0.036.

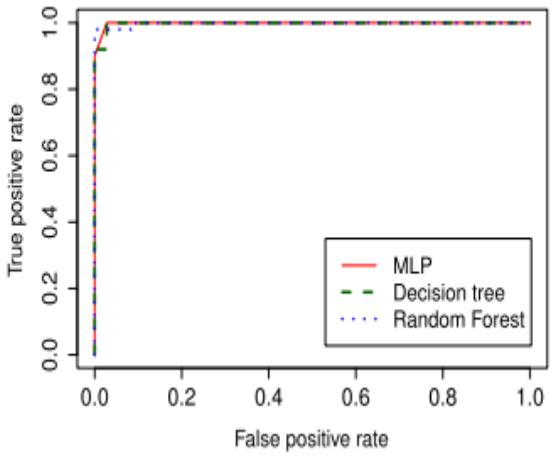

Fig: 8. ROC plot of 3 classifiers: Decision Tree, MLP, and Bagging for app classification (legitimate versus fraudulent). Decision Tree has the highest accuracy, of 98.99 percent. The EER of MLP is as low as 0.01.

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 (CSSD, CSmed, CSmax), the number of reviews with fraud indicator words (fraudW).

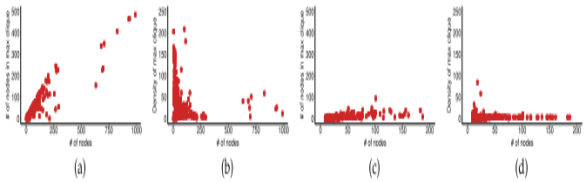

Fig9: Scatterplots for the gold standard fraudulent and malware apps. (a) Each red square represents a fraudulent app, whose y axis value is its number of nodes (reviews) in the largest pseudo-clique identified, and whose x axis value is its number of nodes. (b) For each fraudulent app, the density of 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.

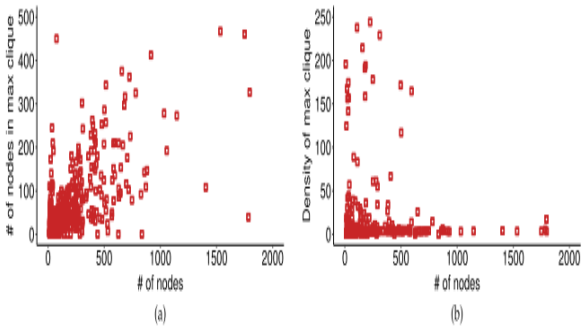

Fig10: Scatterplots of the 372 fraudulent apps out of 1,600 investigated, showing, for each app, (a) the number of nodes (reviews) in the largest clique identified versus the app's number of nodes and (b) the density of the largest clique versus the app's number of nodes. While apps with more nodes also tend to have larger cliques, those cliques tend to have lower densities.

VII.CONCLUSION

In this project, we developed a fraud detection system for mobile Apps. Specifically, we first showed that fraud happened in leading sessions and provided a method for mining leading sessions for each App from its historical ranking records. We identified that for the detection of the rank ranking, rating, review based evidence are considered. Moreover, we proposed an optimization based aggregation method to integrate all the evidence for evaluating the credibility of leading sessions from mobile Apps. A unique perspective of this approach is that all the evidence can be modeled by statistical hypothesis tests, thus it is easy to be extended with other evidence from domain knowledge to detect ranking fraud. Finally, we validate the proposed system with extensive experiments on real-world App data collected from the Apple's App Store. Experimental results showed the effectiveness of the proposed approach. In the future, we plan to study more effective fraud evidence and analyze the latent relationship among rating, review, and rankings. Moreover, we will extend our ranking fraud detection approach with other mobile App related services, such as mobile Apps recommendation, for enhancing user experience

REFERENCES

[1]Alaa Salman Imad H. Elhajj Ali Chehab Ayman Kayss, IEEE Mobile Malware Exposed.International Conference on Knowledge discovery and data mining, KDD'14 pages 978- 983.
[2]Alfonso Munoz, Ignacio Mart ˜ ´ın, Antonio Guzman, Jos ´ e Alberto Hern ´ andez, IEEE Android malware detection from Google Play meta-data: Selection of important features.2015, pages,245-251. [3]Chia-Mei Chen, Je-Ming Lin, Gu-Hsin Lai,IEEE Detecting Mobile Application Malicious Behaviors Based on Data Flow of Source Code.2014 International Conference on Trustworthy Systems and their Applications pp 95-109.
[4]D. F. Gleich and L.-h. Lim. Rank aggregation via nuclear norm minimization. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '11, pages 60–68, 2011.Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Journal of Systems and Software, 2005, in press.
[5]E.-P. Lim, V.-A. Nguyen, N. Jindal, B. Liu, and H. W. Lauw. Detecting product review spammers using rating behaviors. In Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM '10, pages 939–948, 2010.
[6] Andy Greenberg. Malware Apps Spoof Android Market To Infect Phones. Forbes Security, 2014. IEEE Transactions on Knowledge and Data Engineering,Volume:29,Issue:6,Issue Date:June.1.2017 14
[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.