INTERNATIONAL JOURNAL FOR TRENDS IN ENGINEERING & TECHNOLOGY
VOLUME 28 ISSUE 1 – APRIL 2018 - ISSN: 2349 – 9303

Detecting Malware and Finding Ranks in Google Play

Hemalatha .V[1]
PG Scholar, Dept. of CSE
Idhaya Engineering College for Women,Chinnasalem
hemalathavasudha@gmail.com

Deepa .S[2]
PG Scholar, Dept. of CSE
Idhaya Engineering College for Women,Chinnasalem
saideepa1495@gmail.com

Kalaivani .M[3]
PG Scholar, Dept. of CSE
Idhaya Engineering College for Women,Chinnasalem
kalaimoorthy7995@gmail.com

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

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. RELATED WORKS:

System Model- We focus on 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.

Adversarial Model-We consider not only malicious developers, who upload malware, but also have 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 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 on the app on the device. This process complicates the job of fraudsters, who are thus 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.

III. MALWARE DETECTION TECHNIQUES

Signature-based detection:

Signature-based detection works by scanning the contents of computer files and cross-referencing their contents with the "code signatures" belonging to known viruses. A library of known code signatures is updated and refreshed constantly by the anti-virus software vendor. If a viral signature is detected, the software acts to protect the user's system from damage. Suspected files are typically quarantined and/or encrypted in order to render them inoperable and useless. Clearly there will always be new and emerging viruses with their own unique code signatures. So once again, the anti-virus software vendor works constantly to assess and assimilate new signature-based detection data as it becomes available, often in real time so that updates can be pushed out to users immediately and zero-day vulnerabilities can be avoided pattern-marching approach commercial antivirus is an example of signature based malware detection where the scanner scans for a sequence of byte within a program code to identify and report a malicious code. This approach to malware detection adopts a syntactic level of code instructions in order to detect malware by analyzing the code during program compilation. This technique

usually covers complete program code and within a short period of time.

Specification-based malware detection:

Specification based detection makes use of certain rule set of what is considered as normal in order to decide the maliciousness of the program violating the predefined rule set. Thus programs violating the rule set are considered as malicious program. In specification-based malware detection, where a detection algorithm that addresses the deficiency of pattern-matching was developed. This algorithm incorporates instruction semantics to detect malware instances. The approach is highly resilience to common obfuscation techniques. It used template T to describe the malicious behaviors of a malware, which are sequence of instructions represented by variables and symbolic constants. The limitation of this approach is that the attribute of a program cannot be accurately specified. Specification-based detection is the derivate of anomaly based detection.

Behavioral-based Detection:

The behavior-based malware detection system is composed of several applications, which together provide the resources and mechanisms needed to detect malware on the Android platform. Each program has its own specific functionality and purpose in the system and the combination of all of them creates the Behavior-Based malware detection system. The Android data mining script sand applications mentioned in are the responsible for collecting data from Android applications and the script running on the server will be the responsible for parsing and storing all collected data. Furthermore, the script will be responsible for creating the system call vectors for the k-means clustering algorithm.

Cloud Based Malware Detection:

Google Play applications are scanned for malware. Google uses a service named Bouncer to automatically scan applications on the Google Play Store for malware. As soon as an application is uploaded, Bouncer checks it and compares it to other known malware, Trojans, and spyware. Every application is run in a simulated environment to see if it will behave maliciously on an actual device. The applications behaviors compared to the behavior of previous malicious apps to look for red flags. New developer accounts are particularly scrutinized –this is to prevent repeat offenders from creating new accounts Google Play can remotely uninstall applications: If you've installed an app that is later found to be malicious, Google has the ability to remotely uninstall this application from your phone when it's pulled from GooglePlay. Google announced an exciting security feature called the "application verification service" to protect against harmful Android applications. As stated in a recent Google+ post by a member of the Google Android team, "Now, with Jelly Bean Android 4.2 devices that have Google Play installed have the option of using Google as an application verifier. We will check for potentially harmful applications no matter where you reinstalling them from".

IV. THE DATA

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 become active. The add-on interacts 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 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 aloes 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*. 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 parameter for the session context 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, an online malware detector provider, to find out the number of anti-malware tools (Kaspersky, F-Secure) that identify the apk as suspicious. We used 4 servers to collect our datasets.

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 crowd sourcing 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.

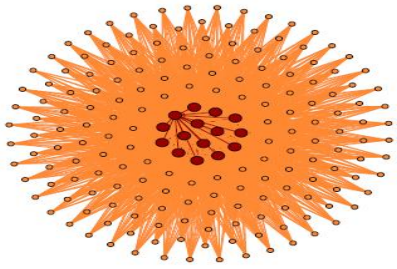

Figure1 Co-review graphs of 15 seed fraud accounts(red nodes) and the 188 GbA accounts (orange nodes)

Gold Standard Data- We used GPad to collect apks of 7756 randomly selected apps from the longitudinal 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" dataset for a total of 212 apps. We collected all the 8,255 reviews of these apps.

Fraudulent Apps- We used contacts established among search rank fraud community, to obtain 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". The review habit 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 201 seed fraud apps (for a total of 6,488 reviews). We call these, *guilt by association* (GbA) accounts. Figure 1 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. 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 extracted from the user who wrote the reviews.

Benign apps- We have selected 925 candidate apps from the longitudinal app set, that have been developed by Google designated "top developers". We have used GPad to filter out those flagged by VirusTotal.

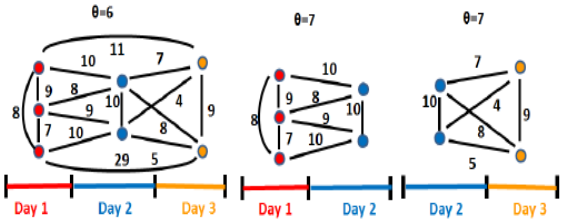

Figure2 (a) (b) (c)

Figure 2: 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) last two days. When θ=8, PCF detects only the clique formed by the first reviews (the red nodes).

V. FAIRPLAY SYSTEM:

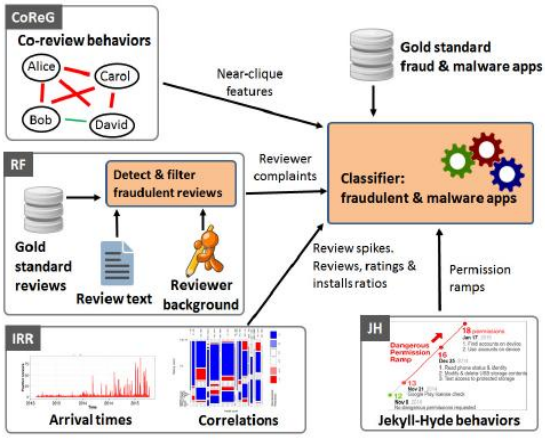

Figure3 Fairplay system architecture

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 Jekyll-Hyde 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.

Genuine reviews- We have manually collected a gold standard set of 315 genuine reviews. 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 author's experience, have at least 150 characters, and are informative.

VI. CLASSIFICATION

Review Classification- The accuracy of Fairplay's fraudulent review detection component (RF module), we used the gold standard datasets of fraudulent and genuine reviews of 3.2. We used GPCrawler to collect the data of the writers of these reviews, including the 203 reviewers of the 406 fraudulent reviews (21, 972 reviews for 2,284 apps) and the 315 reviewers of the genuine reviews (9,468 reviews for 7,116 apps). We observe that the users who post genuine reviews write fewer reviews in total than those who post fraudulent reviews; 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 developers.

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. We evaluate the ability of supervised learning algorithms to correctly classify apps as 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. Finally, we study the most impactful features when classifying fraudulent vs. benign and malware vs. benign apps.

VII. RESULTS

Fairplay has high accuracy and real-world impact.

High Accuracy- Fairplay achieves over 97% in classifying fraudulent and benign apps, and over 95% accuracy in classifying malware and benign apps. FairPlay significantly outperforms the malware indicators. We show that malware often engages in search rank fraud as well: when trained on fraudulent and benign apps, FairPlay flagged as fraudulent more than 75% of the gold standard malware apps.

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 pesudo-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 positive a review for the app, and install and review other apps.

CONCLUSION

In this paper, we propose a novel method to analyze social media data. Our method used K-Means algorithm along with Genetic algorithm and Optimized Cluster Distance method to cluster the social media community based on leadership, follower and attitude scores. With the empirical evaluation, the proposed algorithm outperforms other existing methods. It also presents a use-case of the method to further describe user community by getting more insights from clustering results and assigning self-explanatory labels to each cluster. For future work, the method is to be used with different domains such as Bioinformatics or Image Processing and comparing it with the state-of-the-art methods. We have introduced a FairPlay, a system to detect both fraudulent and malware Google Play apps. 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. 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.

VIII. REFERENCES

[1] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in Proceedings of the ACM Symposium on Foundations of Computer Science, 12-14 Nov. 2000, pp. 359–366.

[2] C. Aggarwal, Data Streams: Models and Algorithms, ser. Advances in Database Systems, Springer, Ed., 2007.

[3] J. Gama, Knowledge Discovery from Data Streams, 1st ed. Chapman & Hall/CRC, 2010.

[4] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," ACM Computing Surveys, vol. 46, no. 1, pp. 13:1–13:31, Jul. 2013.

[5] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in Proceedings of the International Conference on Very Large Data Bases (VLDB '03), 2003, pp. 81–92.

[6] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in Proceedings of the 2006 SIAM International Conference on Data Mining. SIAM, 2006, pp. 328–339.

[7] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, NY, USA: ACM, 2007, pp. 133–142.

[8] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, "Density based clustering of data streams at multiple resolutions," ACM Transactions on Knowledge Discovery from Data, vol. 3, no. 3, pp. 1–28, 2009.

[9] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," ACM Transactions on Knowledge Discovery from Data, vol. 3, no. 3, pp. 1–27, 2009.

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'1996), 1996, pp. 226–231.

[11] A. Hinneburg, E. Hinneburg, and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98). AAAI Press, 1998, pp. 58–65.

[12] L. Ertoz, M. Steinbach, and V. Kumar, "A new shared nearest neighbor clustering algorithm and its applications," in Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining, 2002.

[13] G. Karypis, E.-H. S. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," Computer, vol. 32, no. 8, pp. 68–75, Aug. 1999.

[14] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 3, pp. 515–528, 2003.

[15] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in Proceedings of the International Conference on Very Large Data Bases (VLDB '04), 2004, pp. 852–863.