Search Rank Fraud Detection in Google Play using Co-Review Graph

V. Kavitha
(AP/IT)
Department of Information Technology,
Nandha College of Technology,
Erode, India.

B. Logendhiran, B. Anjitha, S. Nivethini
(UG Scholar),
Department of Information Technology,
Nandha College of Technology,
Erode, India.

Abstract: **The commercial success of Android app markets such as Google Play 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 while malicious developers use app markets as a launch pad for their malware. In this project, 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 in order to identify suspicious apps. Adversaries can have chances to launch attacks by gathering victim's information continuously. This project shows that an adversary can successfully infer a victim's vertex identity and community identity by the knowledge of degrees within a time period. The project also includes a new supervised clustering algorithm to find groups of data (coarse and finer cluster). It directly incorporates the information of sample categories into the fraud clustering process.**

Keywords:-Android market, search rank fraud, malware detection

I. INTRODUCTION

A social networking service (SNS) is a platform to build social networks or social relations among people who share similar interests, activities, backgrounds or real-life connections. A social network service consists of a representation of each user often a profile, his or her social links, and a variety of additional services. Social network sites are web-based services that allow individuals to create

a public profile, create a list of users with whom to share connections, and view and cross the connections within the system. The Most social network services are web-based and provide means for users to interact over the Internet, such as e-mail and instant messaging. Social network sites are varied and they incorporate new information and communication tools such as mobile connectivity, photo, video, sharing. The Online community services are sometimes considered a social network service, though in a broader sense, social network service usually means an individual-centered service whereas online community services are group-centered. Social networking sites allow users to share ideas, pictures, posts, activities, events, and interests with people in their network.

II. LITERATURE SURVEY

CROWDROID: BEHAVIOR-BASED MALWARE DETECTION SYSTEMFOR ANDROID

IKER BURGUERA and URKO ZURUTUZA

The sharp increase in the number of smart phones on the market, with the Android platform posed to becoming a market leader makes the need for malware analysis on this platform an urgent issue. In this paper we capitalize on earlier approaches for dynamic analysis of application behavior as a means for detecting malware in the Android platform. The detector is embedded in a overall framework for collection of traces from a n unlimited number of real users based on crowd sourcing. The method is shown to be an effective means of isolating the malware and alerting the users of a downloaded malware. This shows the potential for avoiding the spreading of a detected malware to a larger community.All market indicators foresee a massive increase in the number of smart phones purchased in the next 5 years. This will create a potential for a massive increase in malware generation, and in particular in the sector dominated by the market leader, potentially the Android platform. In this paper we have proposed a new framework to obtain and analyze smart phone application activity. In collaboration with the Android user's community, it will be capable of distinguishing between benign and malicious applications of the same name and version, detecting anomalous behavior of known applications. Furthermore, by deploying our plat- form on a number of test smart phones, we have created a proof of concept for this mechanism, as a means of analyzing emerging threats. We have indicated that monitoring system calls is a feasible way for detecting malware. This analysis technique has been widely used in the literature. According to the brief survey, we have seen that there're many different approaches to detect malware. We considered that monitoring system calls is one of the most accurate techniques to determine the behavior of Android applications, since they provide detailed low level information. We do realize that API call analysis, information flow tracking or network monitoring techniques can contribute to a deeper analysis of the malware, providing more useful information about malware behavior and more accurate results. On the other hand, more monitoring capability will place a higher demand on the amount of resources consumed in the device. The most important contribution of this work is the mechanism we propose for obtaining real traces of application behavior. We have seen in previous works that it is possible to obtain behavior information using artificially created u ser actions, or creating replicas of smart phones, but crowd sourcing helps the community to obtain real application traces of hundreds or even thousands of applications.

ANDROMALY :A BEHAVIORAL MALWARE DETECTION FRAMEWORK FOR ANDROID DEVICESAUTHORS

ASAF SHABTAI and URI KANONOV

This article presents Andromaly,[a framework for detecting malware on Android mobile devices. The proposed framework realizes a Host-based Malware Detection System that continuously monitors various features and events obtained from the mobile device and then applies Machine Learning anomaly detectors to classify the collected data as normal (benign) or abnormal (malicious). Since no malicious applications are yet available for Android, we developed four malicious applications, and evaluated Andromaly's ability to detect new malware based on samples of known malware. We evaluated several combinations of anomaly detection algorithms, feature selection method and the number of top features in order to find the combination that yields the best performance in detecting new malware on Android. Empirical results suggest that the proposed framework is effective in detecting malware on mobile devices in general and on Android in particular. Since Android has been introduced, it has been (and still is) explored for its inherent security mechanisms, and several targeted security solutions where proposed to augment these mechanisms. From our assessment of Android's security, we believe that additional security mechanisms should be applied to Android. Furthermore, similar to the PC platform, there is no "silver-bullet" when dealing with security. A security suite for mobile devices or smart phones (especially open-source) such as Android includes a collection of tools operating in collaboration. This includes: signature-based anti-virus, firewalling capabilities, better access-control mechanism and also a malware/intrusion detection platform. In this paper we presented a malware detection framework for Android which employs Machine Learning and tested various feature selection methods and classification/anomaly detection algorithms. The detection approach and algorithms are light-weight and run on the device itself. There is however also an option to perform the detection at a centralized location, or at least report the analysis results, derived locally on each device, to such a centralized location.

RISKRANKER: SCALABLE AND ACCURATE ZERO-DAY ANDROID MALWAREDETECTION

MICHAEL GRACE and YAJIN ZHOU

Smart phone sales have recently experienced explosive growth. Their popularity also encourages malware authors to penetrate various mobile marketplaces with malicious applications (or apps). These malicious apps hide in the sheer number of other normal apps, which makes their detection challenging. Existing mobile anti-virus software are inadequate in their reactive nature by relying on known malware samples for signature extraction. In this paper, we propose a proactive scheme to spot zero-day Android malware. With- out relying on malware samples and their signatures, our scheme is motivated to assess potential security risks posed by these untrusted apps. Specifically, we have developed an automated system called Risk Ranker to scalably analyze whether a particular app exhibits dangerous behavior (e.g., launching a root exploit or sending background SMS messages). The output is then used to produce a prioritized list of reduced apps that merit further investigation. When applied to examine 118, 318 total apps collected from various Android markets over September and October 2011, our system takes less than four days to process all of them and effectively reports 3281 risky apps. Among these re- ported apps, we successfully uncovered 718 malware samples (in 29 families) and 322 of them are zero-day. These results demonstrate the efficacy and scalability of Risk Ranker to police Android markets of all stripes. In recent years, smart phones have experienced explosive growth. Gartner reports that worldwide smart phone sales in the third quarter of 2011 reached 115 million units – an increase of 42 percent from the third quarter of the previous year. CNN similarly shows that smart phone shipments have tripled in the past three years. Not surprisingly, multiple smart phone platforms are vying for dominance on these mobile devices. At present, Google's Android plat- form has overtaken Symbian and iOS to become the most popular smart phone platform, being installed on more than half (52 . 5%) of all smart phones shipped. The availability of feature-rich applications (or simply apps) is one of the key selling points that these mobile platforms advertise. By making it convenient for app developers to develop and publish apps, and easy for users to locate and in- stall these apps, platform providers hope to set up a positive feedback loop in which apps will further attract users to their platforms, which in turn drive developers to develop more apps. Various organizations, therefore, have created app stores to facilitate this process. Platform providers tend to offer official distribution services such as Google's Android Market 1 or Apple's App Store. Cellular carriers also provide their own markets and stores, such as AT&T's AppCenter. Moreover, there is third-party markets altogether, ranging from publishing giant Amazon's Appstore to small, specialty markets like Freeware Lovers. Need for market curators to examine or vet apps before accepting them for publication.

USING PROBABILISTIC GENERATIVE MODELS FOR RANKING RISKS OF ANDROID APPS

HAO PENG and CHRIS GATES

One of Android's main defense mechanisms against malicious apps is a risk communication mechanism which, before a user installs an app, warns the user about the permissions the app requires, trusting that the user will make the right decision. This approach has been shown to be ineffective as it presents the risk information of each app in a "stand-alone" fashion and in a way that requires too much technical knowledge and time to distill useful information.We introduce the notion of risk scoring and risk ranking for Android apps, to improve risk communication for Android apps, and identify three desiderata for an effective risk scoring scheme. We propose to use probabilistic generative models for risk scoring schemes, and identify several such models, ranging from the simple Naive Bayes, to advanced hierarchical mixture models. Experimental results conducted using real-world datasets show that probabilistic general models significantly outperform existing approaches, and that Naive Bayes models give a promising risk scoring approach. Android is an open source software stack for mobile devices that includes an operating system, an application framework, and core applications. The operating system relies on a kernel derived from Linux.

ANDROID MALWARE DETECTION USING PARALLEL MACHINE LEARNING CLASSIFIERS

SULEIMAN Y. YERIMA and SAKIR SEZER

Mobile malware has continued to grow at an alarming rate despite on - going mitigation efforts. This has been much more prevalent on Android due to being an open platform that is rapidly overtaking other competing platforms in the mobile smart devices market. Recently, a new generation of Android malware families has emerged with advanced evasion capabilities which make them much more difficult to detect using conventional methods. T his paper proposes and investigates a parallel machine learning based classification approach for early detection of Android malware. Using real malware samples and benign applications, a composite classification model is developed from parallel combination of heterogeneous classifiers. T he empirical evaluation of the model under different combination schemes demonstrates its efficacy and potential to improve detection accuracy. More importantly, by utilizing several classifiers with diverse characteristics, their strengths can be harnessed not only for enhanced Android malware detection but also quicker white box analysis by means of the more interpretable constituent classifiers. An Android application (app) is built from four different types of components: Activities, Services, Broadcast Receivers, and Content Providers. Activities are the components that provide G UI functionality to enable user interactivity, whilst Services and Broadcast Receivers operate in the background when an app is running. Content providers encapsulate data to provide to an app via an interface. Many Android apps consist of at least a number of Activities that are invoked via intents, whilst the other three building blocks may optionally be present depending on the app's functionality. Android apps are written in Java and compiled into a single archive file (Android package or APK), along with data and resource files. Android - powered devices use this APK to install the application. An APK consists of several components including: (1) an XML manifest file containing information such as app descript ion , components declaration (i.e. Activities, Services, Broadcast Receivers etc.), and permissions. (2) A Classes.dex file that is a Dalvik executable file that runs in its own instance of a Dalvik Virtual Machine. (3) A /res directory for indexed resources like icons, images, music etc. (4) A /lib directory for compiled code. (5) /META - INF folder holding the app certificate and list of resources, SHA - 1 digest etc. (6) Resources.arsc which is a compiled resource file.In this paper a parallel classification approach to Android malware detection using inherently diverse machine learning algorithms was investigated. The proposed approach utilized a wide range of features which included API calls related, commands related and permission features. The recent increase in Android malware and their growing ability for adept detection avoidance of existing signature - based approaches definitely calls for novel alternatives.

The parallel classification approach proposed in this paper is a viable scheme that provides a complementary tool that not only potentially improves Android malware detection but al so allows the strengths of diverse classifiers to be leveraged. For example, the rule based classifiers can provide human - interpretable intermediate output that can be useful for driving further analysis stages.learning method in which the weights are obtained from users' behaviors.

GUILT BY ASSOCIATION: LARGE SCALE MALWARE DETECTION BY MINING FILE-RELATION GRAPHS

ACAR TAMERSOY and KEVIN ROUNDY

The increasing sophistication of malicious software calls for new defensive techniques that are harder to evade, and are capable of protecting users against novel threats. We present Aesop, a scalable algorithm that identifies malicious executable files by applying Aesop's moral that "a man is known by the company he keeps." We use a large dataset voluntarily contributed by the members of Norton Community Watch, consisting of partial lists of the files that exist on their machines, to identify close relationships between files that often appear together on machines. Aesop leverages locality-sensitive hashing to measure the strength of these inter-file relationships to construct a graph, on which it performs large scale inference by propagating information from the labeled files (as benign or malicious) to the preponderance of unlabeled files. The outcome of LSH on dataset D is multiple bands, each consisting of a varying number of buckets that contain labeled and unlabeled files. A file appears at most once in a band, inside one of the buckets of the band. Notice that across different bands, the file might appear with a different set of files. For instance, in Table 3, file f 5 appears with file f 4 in two bands and with file f 6 in one band. In this section, we discuss how we combine the buckets from different bands into a unified "structure" and assign labels to unlabeled file s using it.

III. EXISTING SYSTEM

The existing system seeks to identify both malware and search rank fraud subjects in Google Play using FairPlay. This combination is not subjective, it speculate that malicious developers resort to search rank fraud to boost the impact of their malware. The proposed system is built on the observation that fraudulent and malicious behaviours leave behind telltale signs on app markets.FairPlay is a Fraud and Malware Detection Approach which formulate the notion of co-review graphs to model reviewing relations between users. The temporal dimensions of review post times are used to identify suspicious review spikes received by apps. The linguistic and behavioural information is used to detect genuine reviews from which and then extract user-identified fraud and malware indicators. In the existing system, 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) exploits feedback left by genuine reviewers, while the Inter Review Relation (IRR) leverages relations between reviews, ratings and install counts.

IV. PROBLEM DEFINITION

- It does not consider the protection of vertex and community identities of individuals in a dynamic network.
- A privacy model for protecting multi-community identity is not carried out.
- It is identifying both malware and search rank fraud subjects alone not privacy breaches.

V.PROPOSED SYSTEM

The following are the details of proposed system.

- Proposed system includes a new supervised clustering algorithm is proposed to find groups of fraud.
- It directly incorporates the information of sample categories into the fraud clustering process.
- A new quantitative measure is introduced that incorporates the information of sample categories to measure the similarity between users.
- The proposed algorithm is based on measuring the similarity between users using the new quantitative measure.
- So redundancy among the fraud is removed.
- Less dense nodes in the graph is removed so users who rating in minimum amount are not treated as fraudusers.

Advantages

The proposed system has following advantages.

- Number of clusters is prepared and relevance among the fraud is filtered such that coarse as well as finer cluster is prepared.
- Cliques preparation correctly identifies fraud users.
- Densely connected fraud users are also tracked in graphs.

ALGORITHM

A PCF (Pseudo Clique Finder), an algorithm is proposed 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.

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. functionbestNearClique(Graph PC, Set revs)
12. if (PC.size() = 0)
13. for root := 0; root <revs.size(); root++
14. Graph candClique := new Graph ();
15. candClique.addNode (revs[root].getUser());
16. do candNode := getMaxDensityGain(revs);
17. if (density(candClique ∪{candNode} ≥ θ))
18. candClique.addNode(candNode); fi
19. while (candNode != null);
20. if (candClique.density() >maxRho)
21. maxRho := candClique.density();
22. PC := candClique; fi endfor
23. else if (PC.size() > 0)
24. do candNode := getMaxDensityGain(revs);
25. if (density(candClique[candNode) ≥ θ))
26. PC.addNode(candNode); fi
27. while (candNode != null);
28. return.

For each day when the app has received a review (line 1), PCF finds the day's most promising pseudo-clique (lines 3 and 12 -22): start with each review, then greedily add other reviews to a candidate pseudo-clique; keep the pseudo clique (of the day) with the highest density. With that "work-in-progress" pseudo-clique, move on to the next day (line 5): greedily add other reviews while the weighted density of the new pseudo-clique equals or exceeds 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 (get Max Density Gain, 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.Fig.1 illustrates the output of PCF for several θ values.

It is noted that if multiple fraudsters target an app in the same day, PCF may detect only the most densely connected pseudo-clique, corresponding to the most prolific fraudster, and miss the lesser dense ones. CoReG Features.CoReG extracts the following features from the output of PCF (see Table 1) (i) the number of cliques whose density equals or exceeds θ, (ii) the maximum, median and standard deviation of the densities of identified pseudo-cliques, (iii) the maximum, median and standard deviation of the node count of identified pseudo-cliques, normalized by n (the app's review count), and (iv) the total number of nodes of the co-review graph that belong to at least one pseudo-clique, normalized by n.

VI. MODULES

The following modules are present in the project.

- Tweets Collection for reviews.
- Co-Review Graph Construction.
- Finding Cliques to get fraud users.
- Remove nodes with edge weights below threshold so normal users are treated as non-fraud users.

(i) Tweets Collection For Reviews

In this module,

- Using twitter package and search twitter function, the tweets are downloaded and preprocessed.
- Stop word removal, punctuation removal, unicode character removal are carried out.
- Key Terms are filtered such that first 50 more occurrence words are taken.
- Then unique users in the tweet are also found out.

(ii) Co-Review Graph Construction

In this module,

- From unique users in the tweet are found out.

- Same Key word present in two topics of two different users are found, then two nodes and one edge is formed in the graph.
- Thus the full graph is constructed. During edge addition, co-occurrence count is also found out and set as edge weight.

(iii) Finding Cliques to Get Fraud Users

In this module,

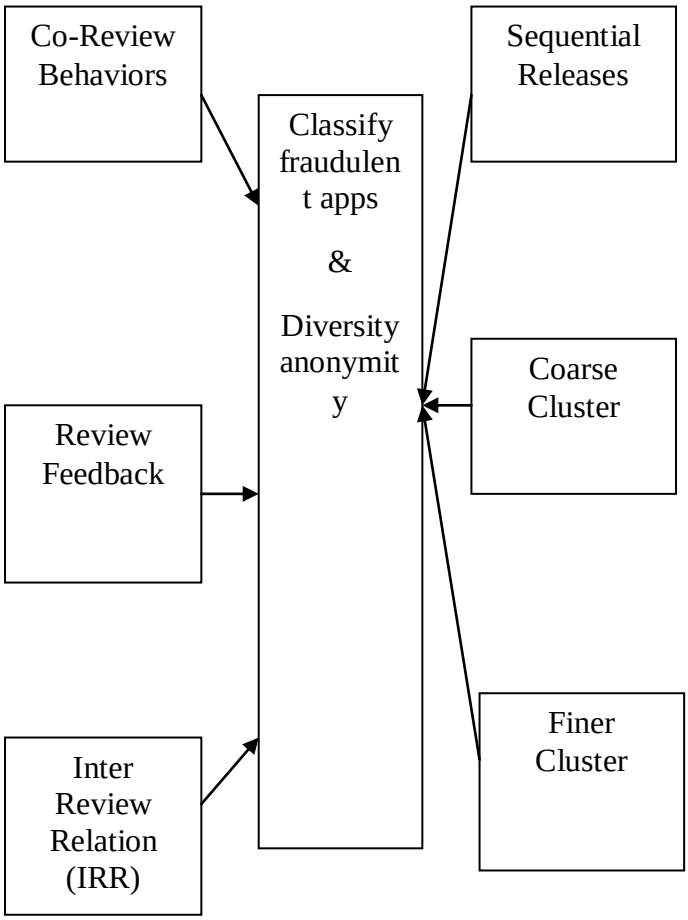
- From the full graph constructed, cliques are found out with minimum 5 nodes in them.
- These cliques denote the users who are densely connected.
- These users are treated as fraud users.

(iv)Remove Nodes with Edge Weights Below Threshold So Normal Users Are Treated As Non-Fraud Users

In this module,

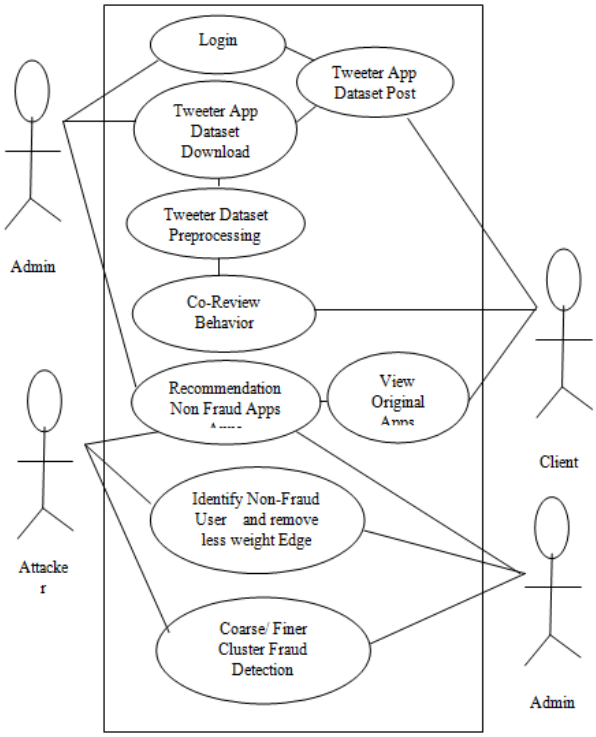
- One nodes, all edges are taken. If all the edge weights are below the given threshold values, it means the user is giving rating less times only.
- The user is treated as normal user.

VII. SYSTEM FLOW DIAGRAM

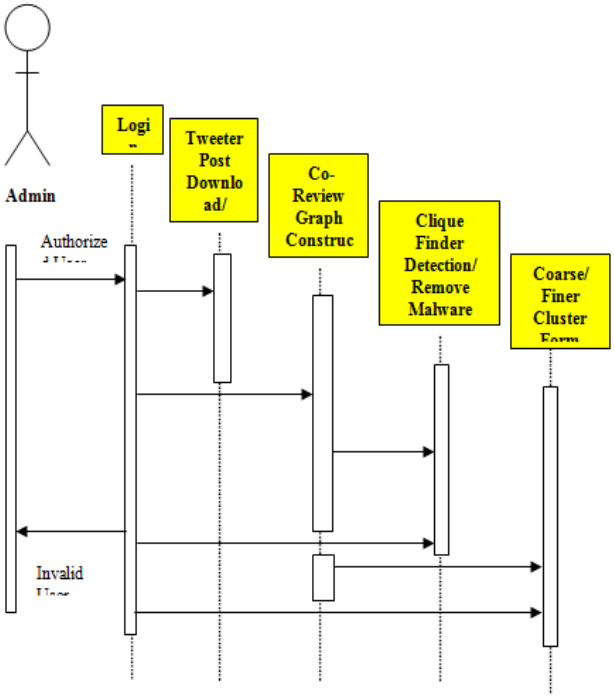


VIII.UML DIAGRAM

(i) Usecase Diagram



(ii) Sequence Diagram



IX. CONCLUSION

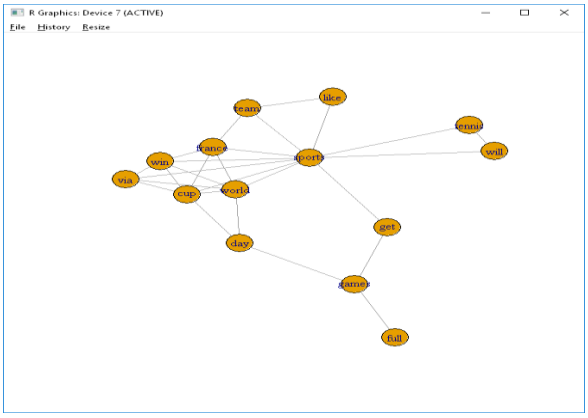
Some fraudulent developers deceptively boost the search rank and popularity of their apps (e.g., through fake reviews and bogus installation counts), while malicious developers use app markets as a launch pad for their malware. The motivation for such behaviors is impact: app popularity surges translate into financial benefits and expedited malware proliferation.

This project seeks to identify both malware and search rank fraud subjects in Google Play. This combination is not arbitrary: we posit that malicious developers resort to search rank fraud to boost the impact of their malware. Unlike existing solutions, this project builds this work on the observation that fraudulent and malicious behaviors leave behind telltale signs on app markets.

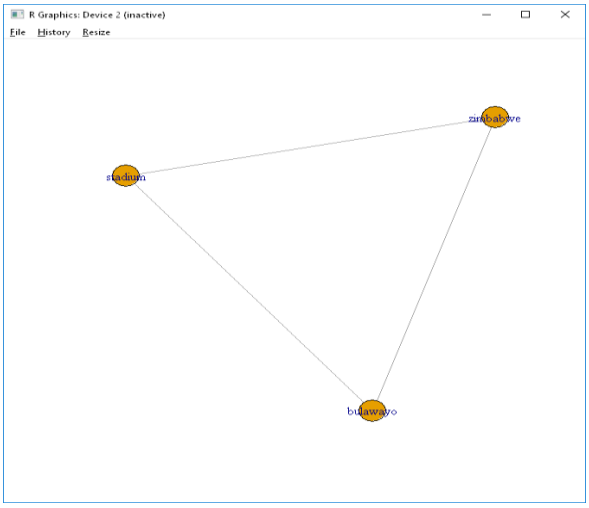
The project has introduced FairPlay, a system to detect both fraudulent and malware Google Play apps. The experiments on the twitter posts, have shown that a high percentage of fraud users are found. In addition, it showed FairPlay's ability to discover non-fraud users also.

X. OUTPUT

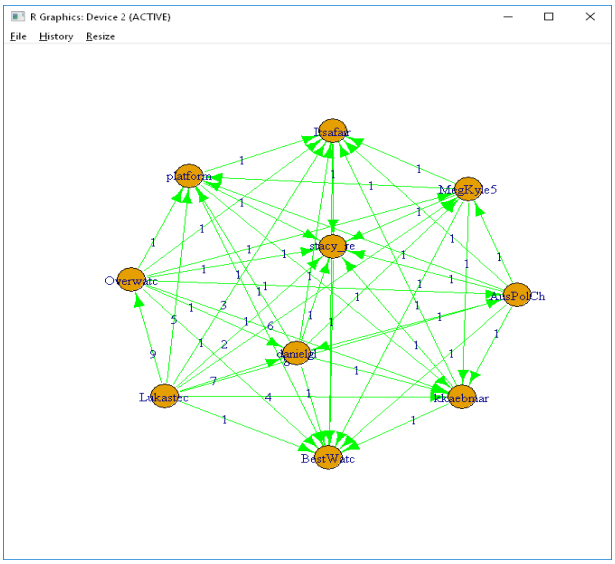
(i) Sample Screen



(ii) CO-REVIEW BACK GROUNDER



(iii) CLIQUE DETECTION



XI. REFERENCES

[1] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani , "Crowdroid: Behavior-based Malware detection system for Android," in Proc. ACM SPSM, 2011, pp. 15–26.

[2] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for Androiddevices, "Intell.Inform. Syst.,vol.38, no.1, pp.161–190,2012.

[3] 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.

[4] 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.

[5] S. Yerima, S. Sezer, and I. Muttik, "Android Malware detection using parallel machine learning classifiers," in Proc. NGMAST, Sep. 2014, pp. 37–42.

[6] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," in Proc. Eur. Intell. Secur. Inf. Conf., 2012, pp. 141–147.

[7] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Alvarez, "Puma: Permission usage to detect malware in android," in Proc. Int. Joint Conf. CISIS12-ICEUTE' 12-SOCO' Special Sessions, 2013, pp. 289–298.