Search Rank Fraud And Malware Detection In Google Play

B. Sudheer Kumar[1], Y. Venkata Ramesh[2] ,Y. Jahnavi[3]
[1]Dept of CSE
[2]Assistant Professor, Dept of CSE
[3]Professor, Dept of CSE
[1, 2, 3] GIST, Gangavaram, Kovur , Nellore

Abstract*- 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 coordinates review activities and uniquely combines detected review relations with linguistic and behavioural signals obtained from Google Play app data in order to identify the suspicious apps. FairPlay achieves over 95% accuracy in classifying gold standard datasets of malware, fraudulent and legitimate apps. We observe that nearly 75% of the malware apps that have been identified are engaged 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.*

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

I. INTRODUCTION

The commercial success of Android app markets such as Google Play [1] and the incentive model they offer to popular apps, make them appealing targets for fraudulent and malicious behaviours. 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 behaviours is impact: app popularity surges translate into financial benefits and expedited malware proliferation.Fraudulent developers frequently exploit crowd sourcing sites to hire teams of willing workers to commit fraud collectively, emulating realistic, spontaneous activities from unrelated people. We call this behaviour "search rank fraud".In addition, the efforts of Android markets to identify and remove malware are not always successful. For instance, Google Play uses the Bouncer system to remove malware. However, out of the 7,756 Google Play apps we analysed using VirusTotal 12% (948) were flagged by at least one anti-virus tool and 2% (150) were identified as malware by at least 10 tools. Previous mobile malware detection work has focused on dynamic analysis of app executables as well as static analysis of code and permissions. However, recent Android malware analysis revealed that malware evolves quickly to bypass anti-virus tools.In this paper, we seek 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, we build this work on the observation that fraudulent and malicious behaviours leave behind telltale signs on app markets. We uncover these nefarious acts by picking out such trails. For instance, the high cost of setting up valid Google Play accounts forces fraudsters to reuse their accounts across review writing jobs, making them likely to review more apps in common than regular users. Resource constraints can compel fraudsters to post reviews within short time intervals. Legitimate users affected by malware may report unpleasant experiences in their reviews. Increases in the number of requested permissions from one version to the next, which we will call "permission ramps", may indicate benign to malware (Jekyll-Hyde) transitions.

1.1 Contributions

FairPlay is a proposed system that supports the above observations to efficiently detect fraud and malware apps in google play. Our major contributions are:

A fraud and malware detection approach. To detect fraud and malware, we propose and generate 28 relational, behavioural and linguistic features, that we use to train supervised learning algorithms [§ 4]:

- Firstly, we should formulate the notion of *co-review graphs* to model reviewing relations between users. We develop PCF, an efficient algorithm to identify temporally constrained, co-review pseudo-cliques — formed by reviewers with substantially overlapping co-reviewing activities across short time windows.

- Linguistic and behavioural type of information can be used for detecting the original reviews and for finding thefraud and malware apps.

Tools to collect and process Google Play data. A tool called GPCrawler, has been developed to automatically collect the data for apps, users and reviews, as well as GPad, which is a tool to download apks of free apps and scan them for malware using VirusTotal.

1.2 Results

FairPlay achieves over 97% accuracy in classifying fraudulent and benign apps, and over 95% accuracy in classifying malware and benign apps. So we can say that it has a higher accuracy rate FairPlay significantly outperforms the malware indicators of Sarma et al. Furthermore, 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 [§ 5.3].

The reviewers of 93.3% of them form at least 1 pseudo-clique, 55% of these apps have at least 33% of their reviewers involved in a pseudo-clique, and the reviews of around 75% of these apps contain at least 20 words indicative of fraud.Through fairPlay we have discovered a novel, *coercive review campaign* attack type, where app users areforcely asked to write a positive review for the app, and install and review other apps. We have discovered 1,024 reviews, from users complaining about 193 such apps [§ 5.4 &§ 5.5].

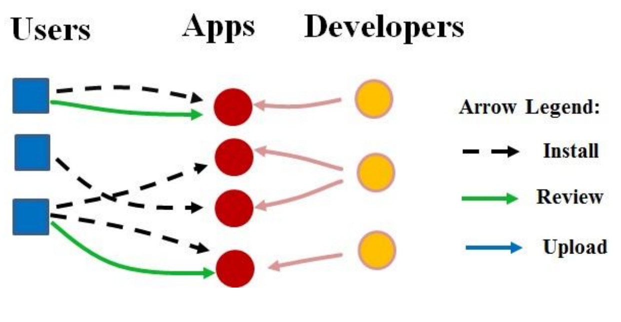


Fig. 2: Google Play components and relations. Google Play's functionality centers on apps, shown as red disks. Developers, shown as orange disks upload apps. A developer may upload multiple apps. Users, shown as blue squares, can install and review apps. A user can only review an app that he previously installed.

II. BACKGROUND, RELATED WORK, AND OUR DIFFERENCES

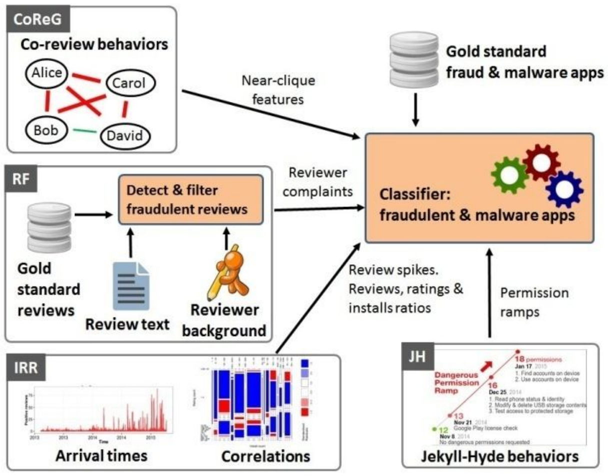
System model. We focus on the Android app market ecosystem of Google Play. The participants, consisting of users and developers, have Google accounts. Developers create and upload apps, that consist of executables (i.e., "apks"), a set of required permissions, and a description. The app market publishes 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. Figure 2 illustrates the participants in Google Play and their relations.To review or rate an app, a user needs to have a Google account, register a mobile device with that account. The reason for search rank fraud attacks is impact. When the search rank of the apps is high such apps will be downloaded mostly. This is beneficial both for fraudulent developers, who increase their revenue, and malicious developers, who increase the impact of their malware.

III. 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. 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 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, G Pad scans each app using VirusTotal 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



FairPlay system architecture. The CoReG module

our datasets, which we describe next. Data

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

Figure 3 shows the distribution of the fresh app categories. 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 2.5stars. However, we observe a spike at more than 8,000 apps with less than one star.

We have collected longitudinal data from these 87,223 apps between October 24, 2014 and May 5, 2015. Specifically, for each app we captured “snapshots” of its Google Play metadata, twice a week. An app snapshot consists of values for all its time varying variables, e.g., the reviews, the rating and install counts, and the set of requested permissions (see § 2 for the complete list). For each of the 2,850,705 reviews we have collected from the 87,223 apps, we recorded the reviewer’s name and id (2,380,708 unique ids), date of review, review title, text, and rating.

This app monitoring process enables us to extract a suite of unique features, that include review, install and permission changes. In particular, we note that this approach enables us to overcome the Google Play limit of 4000 displayed reviews per app: each snapshot will capture only the reviews posted after the previous snapshot.

1.3 Gold Standard Data

Malware apps. We used GPad to collect the apks of 7,756 randomly selected apps from the longitudinal set. Figure 6 shows the distribution of flags raised by VirusTotal, for the 7,756 apks. None of these apps had been filtered by Bouncer 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 Freelancer [7]’s search rank fraud community, to obtain the 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”.

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. 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. Specifically, from this set of 8,457 (= 1,969+6,488) reviews, we have collected 2 reviews from each of the 203 (= 188+15) suspicious user accounts. Thus, the gold standard dataset of fraudulent reviews consists of 406 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 2).

Benign apps. We have selected 925 candidate apps from the longitudinal app set, that have been developed by Google designated "top developers". We have used GPad to filter out those flagged by VirusTotal. We have manually investigated 601 of the remaining apps, and selected a set identifies suspicious, time related co-review behaviours. The RF module uses linguistic tools to detect suspicious behaviours reported by genuine reviews. The IRR module uses behavioural information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.
of 200 apps that (i) have more than 10 reviews and (ii) were developed by reputable media outlets (e.g., NBC, PBS) or have an associated business model (e.g., fitness trackers). We have also collected the 32,022 reviews of these apps. **Genuine reviews**. We have manually collected a gold standard set of 315 genuine reviews, as follows. First, we have collected the reviews written for apps installed on the Android smartphones of the authors. We then used Google's text and reverse image search tools to identify and filter those that plagiarized other reviews or were written from accounts with generic photos. characters, and are informative (e.g., provide information about bugs, crash scenario, version update impact, recent changes).

IV. FAIRPLAY: PROPOSED SOLUTION

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

1.4 FairPlay Overview

FairPlay organizes the analysis of longitudinal app data into the following 4 modules, illustrated in Figure 7. The CoReview Graph (CoReG) module identifies apps reviewed in a contiguous time window by groups of users with significantly overlapping review histories. The Review Feedback (RF) module exploits feedback left by genuine reviewers, while the Inter Review Relation (IRR) module leverages relations between reviews, ratings and install counts. The Jekyll-Hyde (JH) module monitors app permissions, with a focus on dangerous ones, to identify apps that convert from benign to malware. Each module produces several features that are used to train an app classifier. FairPlay also uses general features such as the app's average rating, total number of reviews, ratings and installs, for a total of 28 features.

1.5 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, be a graph where nodes correspond to user accounts who reviewed the app, and undirected edges have a weight that indicates the number of apps reviewed in common by the edge's endpoint users. Figure 16a shows the co-review clique of one of the seed fraud apps (see § 3.2). The co-review graph concept naturally identifies user accounts with significant past review activities.

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 clique if 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.
Maximal clique enumeration algorithms such as [27], [28] applied to co-review graphs are not ideal to solve the problem of identifying sub-sets of an app's reviewers with significant past common reviews. First, fraudsters may not consistently use (or may even purposefully avoid using) all their accounts across all fraud jobs that they perform. In addition, Google Play provides incomplete information (up to 4,000 reviews per app, may also detect and filter fraud). Since edge information may be incomplete, original cliques may now also be incomplete. To address this problem, we "relax" the clique requirement and focus instead of pseudocliques:

The weighted pseudo-clique enumeration problem. For a graph G = (V,E) and a threshold value θ, we say that a vertex sub-set U (and its induced sub-graph G[U]) is a

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}) ≥ $\theta$))
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) ≥ $\theta$))
26. PC.addNode(candNode); fi
27. while (candNode != null);
28. return
```

____

pseudo-clique of G if its weighted density $\rho = \frac{\sum_{e \in E} w(e)}{\binom{n}{2}}$ [29] exceeds θ; n = |V | 1. U is a maximal pseudo-clique if in addition, no other pseudo-clique of G contains U. The weighted pseudo-clique enumeration problem outputs all the vertex sets of V whose induced subgraphs are weighted pseudo-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 θ. PCF outputs a set of identified pseudo-cliques with $\rho \geq \theta$, that were formed during contiguous time frames. In Section 5.3 we discuss the choice of θ. 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 "workin-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 θ (lines 6 and 23 − 27). When no new nodes have been added to the work-in-progress pseudo-clique (line 8), we add the pseudoclique 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-inprogress pseudo-clique, whose writer has written the most apps in common with reviewers already in the pseudoclique. Figure 8 illustrates the output of PCF for several θ values.
If *d* is the number of days over which *A* has received reviews and *r* is the maximum number of reviews received in a day, PCF's complexity is $O(dr^2(r + d))$.

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*.

1.6 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.

1.7 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, **Claim 1**An attacker needs to post at least $\frac{R_A - 1}{5 - R_A}$ positive reviews., in order to counteract for the 1 star review

Proof: Let σ be the sum of all the k reviews received by a before time T. Then, $R_A = \frac{\sigma}{k}$. Let q_r be the number of fraudulent reviews received by A. To compensate for the 1-

starreview posted at time T, qr is minimized when all those reviews are 5 star. We then have that: $R_A = \frac{\sigma}{k} = \frac{\sigma+1+5q_r}{k+1+q_r}$. The numerator of the last fraction denotes the sum of all the ratings received by A after time T and the denominator is the total number of reviews. Rewriting the last equality, we
Rating vs Install count

obtain that $q_r = \frac{\sigma-k}{5k-\sigma} = \frac{R_A-1}{5-R_A}$. The last equality follows by dividing both the numerator and denominator by k. □
Such a "compensatory" behaviour is likely to lead to doubtful high numbers of positive reviews. We detect such behaviours by identifying outliers in the number of daily positive reviews received by an app. Figure 9 shows the timelines and suspicious spikes of positive reviews for 2 apps from the fraudulent app dataset (see Section 3.2).

Reviews, ratings and install counts. To investigate relationships between the install count and the rating count at the end of the collection interval., we used the Pearson's χ2 test, as well as between the install count and the average app rating of the 87K new apps. As Google Play's install count buckets, we grouped the rating count in buckets of the same size. Figure 10 shows the merimakko diagrams of the relationships between rating and install counts. p=0.0008924, thus we result that dependence between the rating and install counts. The cells(rectangles) are identified by the standardized residuals dangerous permissions.

IRR features. We add temporal features (see Table 1): the number of days with detected spikes and the maximum amplitude of a spike. We also take out (i) the ratio of installs to the ratings as two features, I_1/Rt_1 and I_2/Rt_2 and (ii) the ratio of installs to reviews, as I_1/Rv_1 and I_2/Rv_2. $(I_1,I_2]$ denotes the install count interval of an app, $(Rt_1,Rt_2]$ its rating interval and $(Rv_1,Rv_2]$ its (genuine) review interval.
the app's number of dangerous permission ramps, and (iv) its total number of dangerous permissions added over all the ramps.

V. CONCLUSION

We have proposed Fair Play which is a system to detect both fraudulent and malicious software Google Play apps. These experiments on a newly introduced longitudinal app dataset, will be shown that a high percentage of malicious software is involved in search rank fraud; both are accurately identified by Fair Play. Including a new type of coercive fraud attack, these in addition we showed Fair Play's ability to discover hundreds of apps that evade Google Play's detection technology.

REFERENCES

[1] Google Play. https://play.google.com/.
[2] Ezra Siegel. Fake Reviews in Google Play and Apple App Store. Appentive, 2014.
[3] Zach Miners. Report: Malware-infected Android apps spike in the Google Play store. PCWorld, 2014.
[4] Stephanie Mlot. Top Android App a Scam, Pulled From Google Play. PCMag, 2014.
[5] Daniel Roberts. How to spot fake apps on the Google Play store. Fortune, 2015.
[6] Andy Greenberg. Malware Apps Spoof Android Market To Infect Phones.
[7] Forbes Security, 2014.
[8] Freelancer. http://www.freelancer.com.
[9] Fiverr. https://www.fiverr.com/.
[10] BestAppPromotion. www.bestreviewapp.com/.
[11] 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.
[12] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. 3111:260–272, 2004.
[13] Takeaki Uno. An efficient algorithm for enumerating pseudo cliques. In Proceedings of ISAAC, 2007.
[14] Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python. O'Reilly, 2009.
[15] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs Up? Sentiment Classification Using Machine Learning Techniques. In Proceedings of EMNLP, 2002.
[16] John H. McDonald. Handbook of Biological Statistics. Sparky House Publishing, second edition, 2009.
New Google Play Store greatly simplifies permissions. http://www.androidcentral.com/
[17] new-google-play-store-4820-greatly-simplifies-permissions, 2014.
Weka. http://www.cs.waikato.ac.nz/ml/weka/.
[18] S. I. Gallant. Perceptron-based learning algorithms. Trans. Neur.
Netw., 1(2):179–191, June 1990.
[19] Leo Breiman. Random Forests. Machine Learning, 45:5–32, 2001.
[20] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In Proceedings of IJCAI, 1995.
[21] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In Proceedings of the SIAM SDM, 2011.