Journal of Information Processing Vol.26 212–223 (Feb. 2018)

[DOI: 10.2197/ipsjjip.26.212]

Recommended Paper

PADetective: A Systematic Approach to Automate Detection of Promotional Attackers in Mobile App Store

Bo Sun[1,a)] Xiapu Luo[3,b)] Mitsuaki Akiyama[2,c)] Takuya Watanabe[2,d)] Tatsuya Mori[1,e)]

Received: April 29, 2017, Accepted: November 7, 2017

Abstract: Mobile app stores, such as Google Play, play a vital role in the ecosystem of mobile device software distribution platforms. When users find an app of interest, they can acquire useful data from the app store to inform their decision regarding whether to install the app. This data includes ratings, reviews, number of installs, and the category of the app. The ratings and reviews are the *user-generated content* (UGC) that affect the reputation of an app. Therefore, *miscreants* can leverage such channels to conduct *promotional attacks*; for example, a miscreant may promote a malicious app by endowing it with a good reputation via fake ratings and reviews to encourage would-be victims to install the app. In this study, we have developed a system called *PADetective* that detects miscreants who are likely to be conducting promotional attacks. Using a 1723-entry labeled dataset, we demonstrate that the true positive rate of detection model is 90%, with a false positive rate of 5.8%. We then applied our system to an unlabeled dataset of 57 M reviews written by 20 M users for 1 M apps to characterize the prevalence of threats in the wild. The PADetective system detected 289 K reviewers as potential PA attackers. The detected potential PA attackers posted reviews to 136 K apps, which included 21 K malicious apps. We also report that our system can be used to identify potentially malicious apps that have not been detected by anti-virus checkers.

Keywords: mobile app store, promotional attack, machine learning

1. Introduction

With more than four million apps [1], mobile app markets, such as Google Play and Apple App Store, play a vital role in distributing apps to customers. To help users look for apps and for developers to promote their apps, mobile app markets provide a variety of information about the apps, such as descriptions, screenshots, and number of installations. In addition, most markets involve *reputation systems*, through which users can rate the apps and write down reviews, to facilitate other users to select apps. Since apps with higher ratings usually get more downloads [2], recent studies report that some developers adopt unfair approaches to manipulate their apps' ratings and reviews [3], [4], even if such behaviors are prohibited by FTC [5] and app markets. Note that attackers also employ such an approach to promote malicious apps and lure victims to install them. We call such malicious apps campaign as *promotional attacks* (PAs).

Although a few recent studies have revealed the paid reviews [3] and colluded reviewers [4], there have been no systematic examinations on the promotional attacks in mobile app stores. To fill in the gaps, we conducted the first large-scale investigation on PAs with the aim of answering the following two questions: (1) How can we detect PAs systematically? and (2) How prevalent are PAs in the wild?

It is non-trivial to address these two questions because the solution should be accurate to capture PA attackers with low false positive rate, scalable to quickly handle millions of apps and reviews in app stores, and robust to raise the bar for sophisticated attackers to evade the detection. Existing studies cannot achieve these goals. For example, high computational complexity limits the scalability of Ref. [3], and requiring the similar reviews in keyword level affects the accuracy of Refs. [6], [7]. Moreover, to our best knowledge, none of the existing studies have examined market-scale apps.

To tackle these challenges, we propose and develop a novel system, named *PADetective*, to identify PA attackers accurately and efficiently. PADetective adopts supervised learning to characterize PA attackers according to 15 features (e.g., day intervals, semantic similarity), and then applies the trained model to detect other PA attackers. It is worth noting that these new and effective features are carefully selected from not only UGC but also metadata in order to enhance the robustness of PADetective. In particular, features from metadata have not been used by existing works, and they could contribute to the robustness of PADetective because it is easier for attackers to manipulate UGC than metadata. We employ the information entropy and the coefficient of variation for quantifying the features from metadata, and leverage

[1] Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] NTT Secure Platform Laboratories, Musashino, Tokyo 180–8585, Japan
[3] The Hong Kong Polytechnic University, Hong Kong, China
[a)] sunshine@nsl.cs.waseda.ac.jp
[b)] csxluo@comp.polyu.edu.hk
[c)] akiyama.mitsuaki@lab.ntt.co.jp
[d)] watanabe@nsl.cs.waseda.ac.jp
[e)] mori@nsl.cs.waseda.ac.jp

The preliminary version of this paper was presented at Computer Security Symposium 2016 (CSS2016) in October 2016, and recommended to be submitted to Journal of Information Processing (JIP) by the program chair of CSS2016.

the state-of-the-art NLP technique (i.e., *Paragraph vector* [8]) to extract features from UGC because it can extract similar reviews at semantic level and therefore increase the accuracy. Moreover, we employ the TRUE-REPUTATION [9] algorithm to calculate the true reputation scores for detecting abnormal ratings. These algorithms are lightweight, and we only need to recompute the true reputation scores and similarity word weight vectors for new UGC and metadata. This feature extraction approach empowers PADetective to handle large-scale dataset. In our evaluation, PADetective processed 57 million reviews in one day. We evaluate PADetective using real PA data, and the result shows that PADetective's true positive rate is up to 90% with a low false positive rate of 5.8%.

Moreover, we conduct the first large-scale investigation on PA by applying PADetective to 1 million apps in Google Play, which has 57 million reviews posted by 14 million users. PADetective flagged 289 K reviewers as suspicious promotional attackers. These reviewers posted reviews to 136 K apps, which included 21 K malicious apps. Among the top 1 K reviewers who were flagged as promotional attackers with high probability score, 136 reviewers posted reviews only for malicious apps, and another 113 reviewers posted reviews for apps where more than half of the apps were detected as malicious. It is worth noting that PAs detected by PADetective can contribute to the detection of potentially malicious apps.

Major contributions of this work are summarized as follows:

- We developed a novel system, named *PADetective*, which aims to detect PA attackers from a large volume of reviewers with high accuracy and low false positive rates. The extensive experiments demonstrated that PADetective can achieve 90% true positive rate with low false positive rate of 5.8%.
- Using the PADetective, we conducted the first large-scale measurement study on PAs by examining 57 million reviews, posted by 14 million users for 1 million apps in Google Play, and obtained interesting observations and insights.
- Our extensive analyses revealed that the detected PAs can be used to discover potentially malicious apps, which have not been detected by popular anti-virus scanners.

We believe that this research sheds a new light on the analysis of UGC and metadata of app stores as a complementary channel to find malicious apps for enhancing the widely used anti-malware tools or for market operators and malware analysts.

The remainder of this paper is organized as follows. We specify our problem in Section 2. We describe the high-level overview and details of the PADetective in Section 3. A performance evaluation of the PADetective is given in Section 4. In Section 5, we study the promotional attackers in the wild, by applying PADetective to a market-scale measurement data. Section 6 discusses the limitation and future work of our system. Section 7 summarizes the related work and compare them with ours. Finally, conclusions are presented in Section 8.

2. Problem Statement

This section aims to specify the problem we are addressing in this paper. We first present the high-level overview of our problem using a model that represents the user feedback system commonly adopted in the mobile app distribution platforms. We then define our problem in the mathematical way.

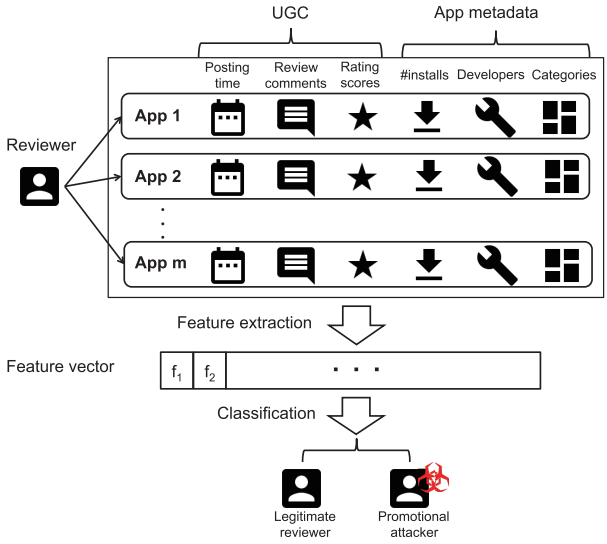

Fig. 1 High-level overview of the problem.

Figure 1 presents the high-level overview of the problem. We note that although this work targets Google Play as an example of mobile app distribution platforms, the model is applicable to other platforms as well. In the model, a reviewer posts review comments and rating scores for several apps published in the app store. For the apps commented/rated by the reviewer, we can extract the UGC and the metadata associated with the apps. The UGC includes comment posting time, review comment, and rating score; these are generated by the reviewer. The app metadata includes the number of installs, a set of developers of the app, and a set of the categories of the app; these are the data of the apps commented/rated by the reviewer.

We now turn our attention to the problem we are addressing in the paper. For a given reviewer, we first compile the UGC and app metadata; we then extract a feature vector from the compiled data. Our goal is to classify the reviewer into two classes: a legitimate reviewer and a promotional attacker. To this end, we apply a supervised machine learning algorithm to the extracted feature vector. In summary, our problem is to determine whether a given reviewer is a promotional attacker or not by analyzing the UGC and the metadata associated with apps commented on or rated by the reviewer.

To formulate the problem in a mathematical way, we introduce the variables summarized in **Table 1**. We note that we only examine the reviewers with $m_i \geq 3$ because it takes time and efforts for promotional attackers to create zombie accounts for commenting apps and therefore they often reuse these accounts for posting reviews. We discuss how to relax this restriction in Section 6. Of the valuables shown in Table 1, c_{ij}, s_{ij}, and t_{ij} are UGC data and n_{ij}, d_{ij}, and k_{ij} are the metadata. Using these six values for all the apps in $\mathbf{A}(r_i)$, we compute a feature vector $\mathbf{F}(r_i) = \{f_1^i, f_2^i, \ldots f_{15}^i\}$ for a given reviewer r_i. Our goal is to build an accurate classifier $g(\mathbf{F}(r_i))$ that determines whether r_i is promotional attacker or not. The details of computing a feature vector from the observed variables will be described in the next section.

Table 1 Notations used for our problem.

Symbol	Definition
r_i	the i-th reviewer ($i = 1, 2, \ldots$)
$\mathbf{A}(r_i)$	a set of apps reviewed by the reviewer r_i.
m_i	number of apps reviewed by the reviewer r_i. $m_i = \lvert\mathbf{A}(r_i)\rvert$.
c_{ij}	review comment posted by the reviewer r_i for the j-th app. $j = 1, 2, \ldots, m_i$.
s_{ij}	rating score posted by the reviewer r_i for the j-th app. $j = 1, 2, \ldots, m_i$.
t_{ij}	time at whch the reviewer r_i posted a comment for the j-th app. $j = 1, 2, \ldots, m_i$.
n_{ij}	number of installs for the j-th app reviewed by the reviewer r_i. $j = 1, 2, \ldots, m_i$.
d_{ij}	developer of the j-th app reviewed by the reviewer r_i. $j = 1, 2, \ldots, m_i$.
k_{ij}	category of the j-th app reviewed by the reviewer r_i. $j = 1, 2, \ldots, m_i$.

3. PADetective System

In this section, we first provide an overview of PADetective, and then detail its four core components: data collection, data preprocessing, feature extraction, and detection.

3.1 Overview

Figure 2 presents an overview of PADetective, which consists of four core components. First, the data collection component has a crawler to collect data from the Google Play Store. Second, the data pre-processing component involves 8 steps in removing noisy data. Third, the feature extraction component obtains the values for the 15 new features from the pre-processed data. Fourth, the detection component selects the most suitable detection model to predict promotional attackers and to determine the correlation between promotional attackers and malicious apps.

3.2 Data Collection

We first create a list of apps to be downloaded by using the list of package names provided with PlayDrone [10]. Then, we collect metadata for each app by accessing its description page according to its package name and employing our HTML parser to extract all metadata in the page. The UGC cannot be obtained from the page directly because listing them involves asynchronous communication with the server. To address this issue, we developed a UGC crawler based on the review collection API [11] provided by Google Play Store. More precisely, our crawler sends an HTTP request, which contains the package name and the page index as parameters, to the server and then parses the JSON file in the HTTP response. **Figure 3** shows the statistics of the number of reviews in each app. We note that the Google Play review collection service only allows 4,500 most recent reviews to be crawled for each app. We could fetch the reviews continuously for circumventing this limitation, thanks to our automated process of data collection. To follow the acceptable use policy of the API, we deployed our crawler on 100 servers around the world to collect UGC for a large number of apps.

We used the crawler to collect UGC and metadata for 1,058,259 apps from the Google Play app store in November 2015. The data set involved 57,868,301 reviews from 20,211,517 unique users. The statistics for the collected UGC and metadata are presented in **Table 2**.

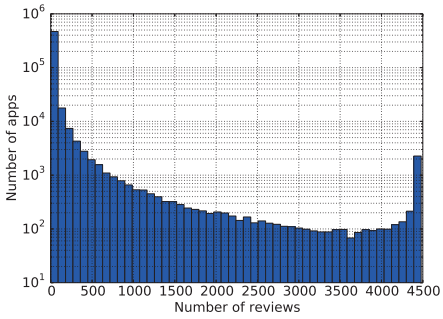

Fig. 3 Histogram for the number of reviews in each app.

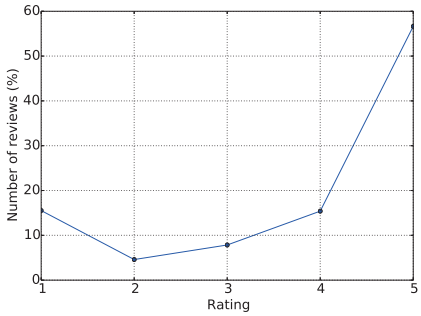

Fig. 4 Percentage of review numbers with different rating.

Figure 4 shows the statistics for the collected rating data. The rating scale in the Google Play Store ranges from 1 to 5. We can see that over 55% of ratings are 5 stars. It may be due to either the users' tendency to give high ratings or PAs. Therefore, it is a challenge to distinguish promotional attackers from legitimate reviewers who are actually satisfied with the apps.

3.3 Data Preprocessing

Before creating the feature vector for the classifier, we develop an 8-step process to remove the noisy and meaningless data.

Step 1: Remove all reviews under the default reviewer name "A Google User," because we cannot extract the string features from the default reviewer name. We discuss how to tackle this limitation in Section 6.

Step 2: Extract the reviewers who have commented on at least three apps. The limitation introduced by this step is discussed in Section 6.

Step 3: Remove reviews written in languages other than English as *PADetective* currently only handles English.

Step 4: Split all sentences into words.

Step 5: Transform all letters into lowercase.

Step 6: Remove all stop words such as "is", "am", "the."

Step 7: Consolidate variant forms of a word into a common form (i.e., word stemming), for example, convert "running" to "run."

Step 8: Correct the misspelling English words for all the reviews.

For Steps 3–8, we implement the natural language processing based on NLTK [12] and TextBlob [13]. NLTK is a widely used Python library for natural language processing, and TextBlob was developed on the basis of NLTK for simplifying text processing. TextBlob enables us to realize language detection and spelling correction in the data preprocessing stage as well as sentiment

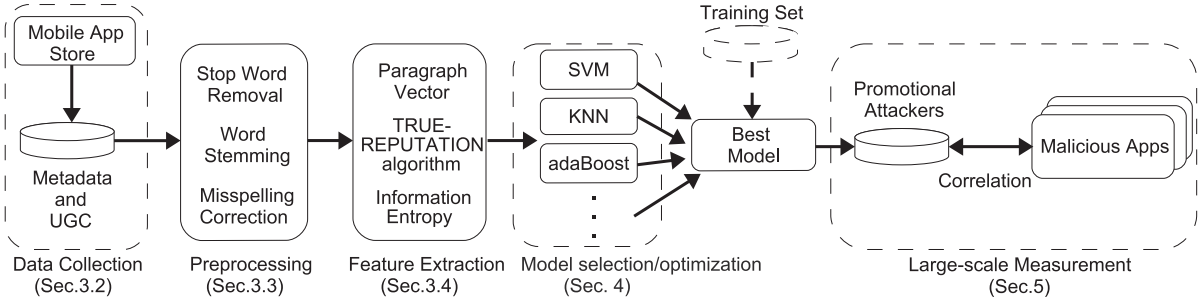

Fig. 2 Overview of PADetective.

Table 2 Description of UGC and metadata.

Type	Item	Description
User-generated content	Reviewer name	The name ID of each reviewer
	Rating	The score attached to each app by reviewer. The range of score is from 1 to 5
	Post time	The date of review creation
	Review	The comment text written by reviewer
Metadata	Number of installs	The count of app downloaded by mobile users, i.e., 1,000–5,000, 10,000+
	Category	The cluster name of apps with similar function, i.e., Entertainment, Communication, Sports
	Developer	The name of an individual or a company who creates the app

analysis during the feature extraction stage. After the 8 steps, 23,255,180 reviews are removed from our dataset. The unique users are only reduced in first 3 steps. The remaining steps are used to preprocess each review by using the natural language processing techniques. In the steps 1, 2, 3, the number of distinct users are reduced to 14,191,879, 2,678,217, 2,606,791, respectively.

3.4 Feature Extraction

We profile each reviewer r_i by using 15 features extracted from the UGC and metadata. These features form a feature vector $\mathbf{F}(r_i) = \{f_1^i, f_2^i, \ldots f_{15}^i\}$, and we classify them into 6 categories, which are detailed in Section 3.4.1–Section 3.4.6.

3.4.1 Posting Time

f_1^i: **Day intervals.** Promotional attackers are likely to launch a rating promotion attack within a short day intervals. For example, Xie and Zhu found that reviewers hired by app promotion web services tend to complete their review promotion missions within 120 days [4]. Therefore, we calculated the day intervals between the earliest and the latest post time $\max(\mathbf{T}_i) - \min(\mathbf{T}_i)$, where $\mathbf{T}_i = \{t_{i1}, \ldots, t_{im_i}\}$, and defined $f_1^i = \max(\mathbf{T}_i) - \min(\mathbf{T}_i)$.

f_2^i: **Day entropy.** Promotional attackers are likely to write reviews within the same day, because they may use automated posting process or want to get paid as quickly as possible. To measure the proportion of same-day reviews, we defined f_2^i using the information entropy as follows:

$$f_2^i = H(X) = -\sum_{j=1}^{m_i} P(t_{ij}) \log P(t_{ij})$$

where $P(t_{ij})$ is the frequency of same-day reviews: $\frac{t_{ij}}{sum}$ and $sum = \sum_{j=1}^{m_i} t_{ij}$ is the sum of days reviewed by reviewer r_i. We note that H (Greek capital letter eta) expresses Shannon entropy. If all the reviews are posted on the same day, the entropy of the post time will be 0.

3.4.2 Reviews

f_3^i: **Bi-gram matching.** Promotional attackers often post similar reviews. Detecting similar reviews is important due to the presence of made-up words that are used to express strong feelings, such as "goooooood" and "coooooool." Made-up words cannot be reformed by existing spelling correction algorithms because they are designed to correct misspelled words instead of intentionally created words. To address this problem, we converted each word into a bi-gram and then used bag of bi-gram to build a feature vector for each c_{ij}. Finally we calculated the average of the cosine similarity score of each pair of reviews by the reviewer r_i. In other words,

$$f_3^i = \frac{\sum_{j=1}^{m_i} \sum_{k=1}^{m_i} cosim(c_{ij}, c_{ik})}{m_i^2}$$

where cosim is cosine similarity score. We set the threshold of cosine similarity as 0.9

f_4^i: **Semantic similarity.** Since reviewers may use different words and expressions to express the same feeling, we identify similar words and expressions using the Paragraph Vector (PV) algorithm [8], because it performs a semantic analysis in discovering similar words and expressions. More precisely, the PV algorithm has each document represented by a dense vector, which is trained by stochastic gradient descent and back-propagation, to predict the similarity of words in the different documents. The PV algorithm is designed in a distributed way such that it can train a large amount of unlabeled data in a very short period of time. For example, by applying the PV algorithm realized in the Python library gensim [14] to 57,868,301 reviews in our dataset, we get the predicted model after around 1 hour. We defined f_4^i as the average of the similarity scores predicted from the trained model for each pair of reviews.

$$f_4^i = \frac{\sum_{j=1}^{m_i} \sum_{k=1}^{m_i} D(c_{ij}, c_{ik})}{m_i^2}$$

Table 3 Examples of similarity score computed with the trained Paragraph vector model.

word1	word2	similarity score
adware	malware	0.88
ads	spam	0.64
camera	permission	0.74
hack	access	0.71
internet	location	0.62
good	nice	0.60

Table 4 Example of score predicted by sentiment analysis classifier.

Sentence	The score of sentiment analysis
That is my opinion	0.0
Awesome game.	0.3
Nice graphics and I love it.	0.55
Very bad game.	−0.65
I hate all the covers I'm here to look for the songs made by the artist not covers.	−0.8

Where D is the distance of two different documents computed by PV algorithm.

Table 3 presents some examples of the similarity scores computed by the trained PV model. It is clear that the model can infer the correlations between not only different words with the same purpose but also security-related similarity words without using the labeled data. Note that although we used words to demonstrate the effectiveness of the approach, we actually apply the algorithm to the entire review texts.

f_5^i: **Sentiment analysis.** Promotional attackers usually post positive reviews to promote apps for monetary benefit and/or luring more victims to install malicious apps. Sentiment analysis is an approach used to classify the feeling of a given text into three categories: negative, neutral, positive. By using the sentiment analysis, we can extract potential promotional attackers who have posted only positive reviews to the apps.

We use TextBlob [13] to conduct the sentiment analysis of all the reviews. The sentiment analysis in TextBlob was implemented by a supervised learning naive bayes classifier that is trained on the labeled movie reviews provided by NLTK. The bag-of-words approach is used for feature vector creation. The accuracy of the sentiment analysis classifier is between 80% and 90%. In our case, both the training data (movie reviews) and predicted data (android app store reviews) were different types of reviews with similar characteristics. Therefore, the sentiment analysis classifier could achieve a high prediction accuracy of 90% for our review data. We defined f_5^i as the average score for each pair of reviews predicted by the sentiment analysis classifier. **Table 4** shows an example of the scores predicted by the sentiment analysis classifier. If the score is zero, it means the sentiment of the review is neutral. We found that our classifier had identified the sentiment of the reviews correctly.

f_6^i: **The average length of the reviews.** Fake reviews injected by promotional attackers are likely to be short, because they may use an automated posting process or want to get income as quickly as possible. Therefore, we defined f_6^i as the average length of the reviews written by the reviewer r_i.

3.4.3 Ratings

f_7^i: **True Reputation Score.** Users often rely on the average ratings of the apps, computed by the app stores, in selecting the apps. Unfortunately, promotional attackers can easily manipulate the average ratings by giving high ratings to their target apps. We defined f_7^i as the average of the margin between the app's rating and the reviewer's rating based on the true reputation score of each app instead of the average rating. This score is calculated according to the TRUE-REPUTATION algorithm [9], which takes into account the user confidence in terms of user activity, user objectivity, and user consistency.

User activity, objectivity, and consistency are in the range of $[0, 1]$. If the activity score of a user v_r is 1.0, the user is the most active user and posts many app ratings. The user objectivity o_r indicates the aggregated objectivity of the ratings posted by a user, with a value of 1.0 indicating the most objective user. The user consistency i_s is used to detect abnormal ratings by applying boxplot analysis. A reviewer with legitimate behavior is given a high user consistency score. After computing these three scores, the confidence of a rating s, u_s, can be calculated as

$$u_s = v_r \times o_r \times i_s, s \in \mathbf{S}_r,$$

where r is a reviewer and $\mathbf{S}_r$ is set of ratings posted by reviewer, r. Finally the true reputation score is defined as

$$u_a = \frac{\sum_{s \in \mathbf{S}_a}(s \times u_s)}{\sum_{s \in \mathbf{S}_a} u_s},$$

where a is an app and $\mathbf{S}_a$ is the set of ratings for app a. Based on u_a, f_7^i is computed as:

$$f_7^i = \frac{\sum_{i=1}^{m_i}(s_{ij} - u_{aj})}{m_i},$$

where m_i is the number of apps reviewed by reviewer r_i.

f_8^i: **Average ratings.** Since promotional attackers will give high ratings to malicious apps for attracting more downloads, we defined f_8^i as the average ratings posted by reviewer r_i.

f_9^i: **Coefficient of variation of ratings.** We defined f_9^i as the coefficient of variation of all the ratings posted by each reviewer to measure their distribution. The coefficient of variation is the ratio of the standard deviation to the mean.

$$f_9^i = \frac{\sigma(\mathbf{S}_i)}{\sum_{j=1}^{m_i} s_{ij}},$$

where σ is standard deviation and $\mathbf{S}_i = \{s_{i1}, \ldots, s_{im_i}\}$. If a reviewer posts identical ratings, the coefficient of variation will be 0.

3.4.4 Number of Installs

f_{10}^i: **Average number of installs.** Since the number of installs is an important metric affecting users' selection of apps, we defined f_{10}^i as the average number of installs for reviewer r_i.

$$f_{10}^i = \frac{\sum_{j=1}^{m_i} n_{ij}}{m_i},$$

f_{11}^i: **Coefficient of variation of the number of installs.** To measure the distribution of the number of installs, we define

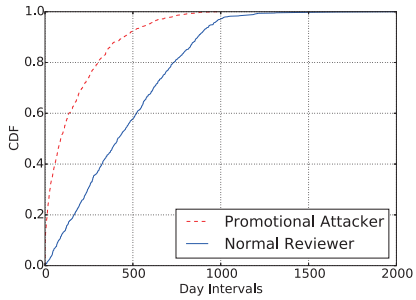

Fig. 5 f_1^i: Day Intervals.

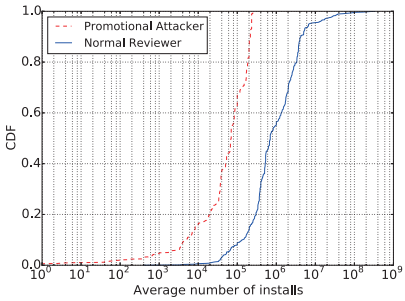

Fig. 6 f_{10}^i: Average number of installs.

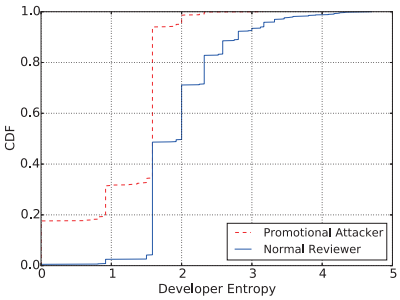

Fig. 7 f_{12}^i: Developer Entropy.

f_{11}^i as the coefficient of variation of the number of installs for reviewer r_i. The computation of f_{11}^i can be referred to the equation defined by f_{10}^i.

If a reviewer posts reviews to apps with the same number of installs, the coefficient of variation will be 0.

3.4.5 Developer and Category

f_{12}^i: **Developer Entropy.** Promotional attackers are more likely to promote apps produced by the same developer because the targeted malicious apps should be associated with each other. Therefore, we defined f_{12}^i as the entropy of developer for reviewer r_i. The computation of f_{12}^i can be referred to the equation defined by f_2^i. If a reviewer only posts reviews for apps from the same developer, the entropy of the developers related to reviewer r_i will be 0.

f_{13}^i: **Category Entropy.** As a promotional attacker may target apps that are popular or in the money-making category such as game category. Similar with f_{12}^i, we defined f_{13}^i as the entropy of category for reviewer r_i. The computation of f_{13}^i can also be referred to the equation defined by f_2^i. If a reviewer only posts reviews for apps having a small number of distinct categories, the entropy of the categories related to reviewer r_i will be 0.

3.4.6 Reviewer names

f_{14}^i: **Length of reviewer name.** Legitimate reviewers usually use their own name as the reviewer name, whereas the reviewer names selected by promotional attackers are likely to be unusually short or long. Hence, we defined f_{14}^i as the length of the reviewer name.

f_{15}^i: **Number of digits and symbols in reviewer name.** The reviewer names of promotional attackers are often randomly generated, and therefore they are likely to contain digits and symbols such as "!", "*", "@." According to this observation, we defined f_{15}^i as the number of digits and symbols in the reviewer names.

3.5 Effectiveness of feature

In the following, we demonstrate how our features work in detecting promotional attackers. In particular, we picked the top-3 feature that contributed most to the classification. The top-3 features are f_1^i: Day intervals, f_{10}^i: Average number of installs, and f_{12}^i: Developer Entropy. f_1^i: Day intervals is the most influential one in these features. We extracted these three features by using a tree-based feature selection method [15], which uses forests of trees to evaluate the importance of features.

Figure 5 shows the Cumulative Distribution Function (CDF) of the day intervals of promotional attackers and those of normal reviewers. We can see that promotional attackers usually have shorter day intervals than normal reviewers. It is likely that promotional attackers want to get revenue quickly or are required by their employers to do so. **Figure 6** shows the CDF of the number of installs of promotional attackers and those of normal reviewers. We can figure out that promotional attackers tend to promote apps whose number of installs is not very large due to the prohibition of promotion activity by Google Play [16]. **Figure 7** shows the CDF of the developer entropy of promotional attackers and those of normal reviewers. We can see that promotional attackers tend to promote apps produced by the same developer. Because promotional attackers are probably hired by the same developer.

We note that these three features are informative for identifying promotional attackers from normal reviewers. We also found that the features extracted from metadata are more effective than those from UGC in PA detection, because it is not easy for attackers to manipulate the metadata such as developer and number of installs.

3.6 Detection

We built our detection model based on the machine-learning algorithms implemented in the Python library scikit-learn [17] because this library is efficient. Considering the performance of each machine learning method, we adopted standard supervised learning methods, i.e., support vector machine (SVM), k-nearest neighbor (KNN), random forest, decision tree, and adaBoost. To determine the best machine-learning algorithm and parameters, we leveraged our labeled dataset to test all the selected models using classifiers and parameters. The detailed model selection process and its results are presented in Section 4. Finally, we applied the best detection model to perform a large-scale analysis of our real-world dataset.

4. Performance Evaluation

This section presents the evaluation result of PADetective. We first introduce how we prepare the labeled dataset (i.e., the *ground truth*), and then describe the evaluation method and the result, respectively.

4.1 Training Dataset

We first generate the training dataset with the ground truth.

Considering that there may be legitimate reviewers who comment on a bad app or post reviews to malicious apps by mistake, we define a promotional attacker as a reviewer who posts reviews only to malicious apps, and the number of malicious apps is at least three apps because it is likely that promotional attackers promote more malicious apps to increase infected devices or monetary benefits quickly. In contrast, legitimate reviewers post reviews only to benign apps.

We determine whether an app was malicious by submitting the app to VirusTotal [18] and making the decision based on the results from a set of antivirus systems. Note that we did not verify all the apps in our dataset to generate the training dataset because of the limitation of time and computer resources. The number of apps we used for collecting UGC and metadata is 1,058,259. After the data preprocessing, 234,139 apps are left for the large-scale analysis. We also note that VirusTotal usually classifies malicious apps into two categories: malware and adware. VirusTotal provides several detection names of a given malware or adware. We can use the names to distinguish between malware and adware. If we observe the names for both malware and adware, we adopt the most frequent types as our choice. We did not distinguish between these categories because PAs would likely be used to promote both malware and adware apps. With this approach and additional manual inspection, we identified 723 promotional attackers. Aside from this, we randomly selected 1,000 legitimate users to create the training dataset. The reason why we randomly sampled legitimate users was to achieve a good balance between the two classes when we trained our classifiers.

4.2 Evaluation Method

Figure 8 shows the flow of performance evaluation. We randomly divided the labeled data into two sets. Containing 70% of labeled data, the first dataset is the training dataset used to optimize each machine learning model and select the best model. For optimizing the machine learning algorithms, we specify a set of carefully chosen values for each parameter in machine learning algorithms. i.e. for random forest, we set parameter "n_estimators" to a set of values: 50, 100, 150, 200, 250. Then we evaluated machine learning algorithms with different parameters by using 10-fold cross-validation. Finally we selected best result in consideration of accuracy, false positive and false negative. Having 30% of labeled data, the second dataset is the test dataset utilized to evaluate PADetective's performance after the best model is selected. Given that we did not use this test set for optimizing/selecting the model, the prediction results for it can be thought as test for the unknown data.

To measure the accuracy of various supervised learning algorithms, we use three metrics: false positive rate (FPR), false negative rate (FNR) and accuracy (ACC), where $FPR = \frac{FP}{FP+TN}$, $FNR = \frac{FN}{TP+FN}$, and $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, respectively. TP is true positive, FP is false positive, TN is true negative and FN is false negative. We also show the performance of the best detection model through the ROC curve, which can be used to determine the best combination of true and false positive rates.

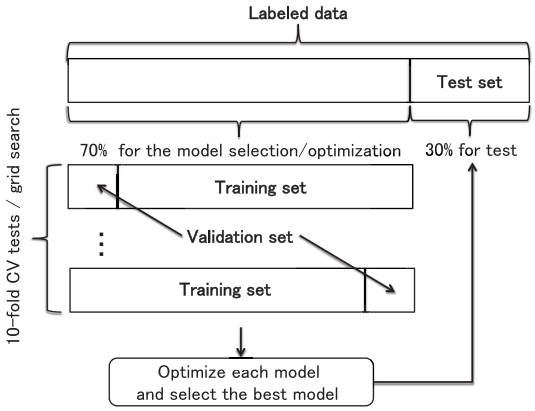

Fig. 8 A flow of evaluating the accuracy of PADetective.

Table 5 Classification accuracy. The means and standard deviations are calculated using 10-times 10-fold cross-validation tests for each machine learning algorithm.

Machine learning	ACC		FPR		FNR	
Algorithm	mean	std	mean	std	mean	std
SVM	0.661	0.041	0.059	0.072	0.372	0.048
RandomForest	**0.933**	**0.014**	**0.083**	**0.033**	**0.053**	**0.036**
KNN	0.894	0.020	0.162	0.027	0.050	0.022
DecisionTrees	0.902	0.020	0.091	0.035	0.100	0.033
AdaBoost	0.918	0.022	0.100	0.030	0.066	0.034

4.3 Evalutaion Result

Table 5 lists the accuracy of different machine learning algorithms used by PADetective. Most of these algorithms predicted the promotional attackers with high accuracy and low false negative or false positive rate. Among the five machine-learning algorithms we tested, RandomForest achieved the highest accuracy (i.e., 93.3%) with the lowest false positive (i.e., 0.083) and false negative (i.e., 0.053) rates. Moreover, its standard deviations of the accuracy, false positive rate, and false negative rate of RandomForest are also low, indicating that RandomForest can identify promotional attackers effectively. We use the grid search to determine the best parameter for RandomForest, and find that 50 is the optimal number of trees. Based on these results, we select RandomForest as our detection model. With regard to the metrics we used, F-measure is one of the useful metrics that can capture the trade-offs between the accuracy and error. We adopted another metric that can also capture this trade-off. As we showed in Table 5, the random forest algorithm achieved the highest accuracy while achieving the lowest FPR. It also achieved the second lowest FNR. Although kNN achieved the lowest FNR, its accuracy and FPR are not better than random forest. Given these observations, we can conclude that Random Forest works the best for our problem. Meanwhile, we optimize all the machine learning algorithms we used. Generally, it is not a straightforward task to identify the reason why one machine learning algorithm works the best. Although not conclusive, we conjecture that the reason why Random Forest works the best in our study might come from the fact that it tends to have less variances [19].

To better understand the root causes of false negative rate and false positive rate in our system, we conduct error analysis with manual inspection. It turns out that PADetective failed to detect the promotional attackers who had posted reviews for a period of two years or longer. On the other hand, PADetective wrongly

Table 6 Statistics of detected promotional attackers and apps. "–" indicates that we were not able to perform the evaluation due to the lack of resources.

	# reviewers	# apps	# malicious apps	# apps deleted by app store
All observed reviewers	2,605,068	234,139	32,367	–
Potential promotional attackers	289,000	135,989	20,906	–
Detected promotional attackers with high confidence	1,000	2,904	486	148

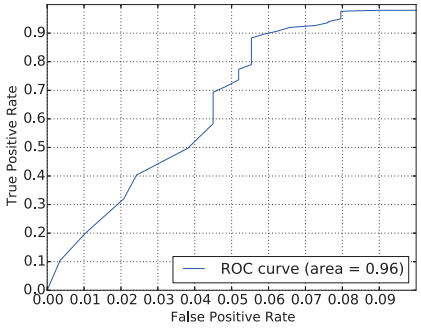

Fig. 9 Evaluation of detection model using test set. Note that we did not use test set to train the classifier.

Table 7 Top-10 types of malicious apps reviewed by the detected PAs.

Type of malicious app	The number of type
Android.RevMobAD.A (AdWare)	18.5%
Adware.Android.Gen	8.4%
Android.Airpush.G (AdWare)	4.7%
Android.Leadbolt.A (AdWare)	3.5%
Trojan.AndroidOS.Generic.A	2.9%
Android.Airpush.H (AdWare)	2.9%
Adware/ANDR.StartApp.A.Gen	2.7%
Adware.AndroidOS.Startapp	2.1%
Riskware.Android.Leadbolt.dkzuxh	1.9%
Adware/ANDR.Leadbolt.B.Gen	1.6%

flagged the legitimate reviewers whose behaviors were similar to a promotional attacker (e.g., their reviews seemed to be fake, but the apps reviewed were not detected by the VirusTotal). It is worth noting that advanced malware may evade the online virus checkers.

Finally, using the optimized RandomForest algorithm, we test PADetective's accuracy using the test dataset. **Figure 9** shows that it can achieve 90% true positive rate with low false positive rate of 5.8%. We can claim that such accuracy is good for the unknown set, indicating that the classification scheme is robust. In the next section, we will use this classification model to investigate the PAs in the large-scale data.

5. Promotional Attacks in the Wild

5.1 Large-scale Measurement

Using PADetective, we conducted a large-scale analysis of real-world data collected from the Google Play Store, and found 289,000 potential promotional attackers from 2,605,068 reviewers. In Section 4.1, we used a 1723-entry labeled dataset (a small portion of all the dataset) to build and test our classifier. In Section 5.1, we conducted a large-scale analysis on the remaining "unlabeled" data by using the classifier we built in Section 4.1. **Table 6** summarizes the number of reviewers/apps detected by PADetective. The number of unique malicious apps reviewed by the potential promotional attackers was 20,906, accounting for approximately 65% of the malicious apps reviewed by all observed reviewers. It is worth noting that many malicious apps having reviews were associated with the potential promotional attackers. Note that the majority of malicious apps detected by VirusTotal had no user reviews. It is likely that malicious apps are detected and deleted by mobile app stores in the early stage of distribution, so there are no users to use and comment on such malicious apps. Another possibility is that mobile app stores deleted both malicious apps and their information including reviews simultaneously, so we can not collect the reviews from mobile app store.

Then, we ranked the reviewers in descending order according to the probability of being a promotional attacker, and investigated the top 1,000 reviewers detected as promotional attackers. The top 1,000 reviewers posted reviews for 2,904 of apps, which include 486 of malicious apps and 148 of apps deleted by the app store for some reasons, e.g., malware or potentially harmful apps. Among the 486 malicious apps, approximately 66% of malicious apps are labeled as adware. We present the top 10 types of malicious apps that are reviewed by the detected promotional attackers in **Table 7**. Most of them are also labeled as adware. Promotional attackers tend to promote adware so that they can make more profiles and collect users' information.

Among the 1,000 promotional attackers, 136 reviewers (13.6%) posted reviews only for malicious apps or the deleted apps. We found that other detected reviewers posted reviews for not only malicious apps, but also for apps that were not regarded as malware/adware by VirusTotal. We note that using the online virus checkers could be one of the sources of false detection. We leave the checking of the code of those undetected apps for our future work.

Figure 10 shows the top 10 categories of the apps reviewed by promotional attackers. Three categories (approximately 15% in total) are related to game, which was the primary target of the PAs. To study the impact of apps promoted by the promotional attackers, **Fig. 11** illustrates the top 10 number of installs of the apps reviewed by promotional attackers. We can see that the majority of such apps do not have many installs. This observation indicates that PAs are used when the app is not so popular. There may be other reasons that the data was captured when the PA was just launched (i.e., not yet finished),

We also investigate whether the detected promotional attackers can be used to discover malicious apps. More precisely, we compare the time when the promotional attackers posted reviews on malicious apps and the time when the malicious app was first submitted to VirusTotal. If all the posting times are earlier than the first submission time, then our PA detection scheme has the potential to identify new, previously unknown malicious apps soon after publication.

Table 8 A set of apps reviewed by a detected promotional attacker.

App Name	Reviews	Ratings	True Reputation Score	Post Time	Category	Developer	Downloads	VirusTotal Detection
com.wb.atones	Great. Application	5	3.71	2013.12.20	MUSIC AND AUDIO	Navjot Singh	10,000+	*ANDR.RevMob.A*
com.wb.bankpo	Great app for. Preparing banking exams.	5	4.05	2013.12.20	EDUCATION	Navjot Singh	5,000+	*Android.RevMobAD.A*
com.wb.bhangra	Great boliyan	5	2.85	2013.12.20	MUSIC AND AUDIO	Navjot Singh	10,000+	*ANDR.RevMob.A.Gen*
com.wb.dbreed	Great dogs. Name	5	3.88	2013.12.20	EDUCATION	Navjot Singh	1,000+	*Android.RevMobAD.A*
com.wb.htones	Awesome horror tones.	5	3.07	2013.12.20	MUSIC AND AUDIO	Navjot Singh	10,000+	*ANDR.RevMob.A*
com.wb.piczzle	Piczzle app great application. It is a awesome app	5	4.54	2013.12.20	GAME PUZZLE	Navjot Singh	500+	*Trojan.AndroidOS. Generic.A*
com.wb.sukhmani	Waheguru wahe-guru	5	4.34	2013.12.20	EDUCATION	Navjot Singh	10,000+	*Trojan.AndroidOS. Generic.A*

Table 9 A set of unknown malicious apps reviewed by a detected promotional attacker.

App Name	Reviews	Ratings	Post Time	Category	Developer	Downloads	VirusTotal Detection	First submission date on VirusTotal
com.ArabicAlphabets .memory.mathes	Nice	5	2014.02.03	GAME PUZZLE	GameLab	5,000+	*AndroidOS/GenBl. F33291AF!Olympus*	2014.08.03
com.electricity.billinf o.free	Nice	5	2014.02.03	ENTERTAINMENT	GameLab	5,000+	*AndroidOS/GenPua. 14444BDA! Olympus*	2014.07.30
com.siminfo.gsm.free	Good	5	2014.02.03	ENTERTAINMENT	GameLab	5,000+	*AndroidOS/GenPua. A0CB31EE! Olympus*	2014.07.30

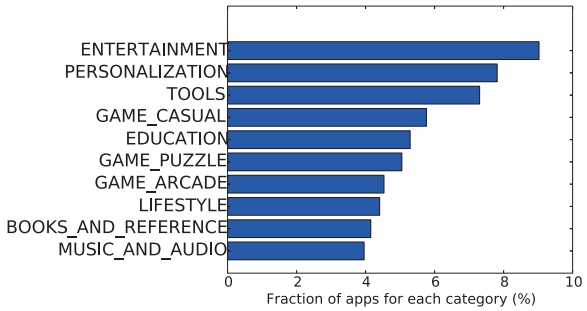

Fig. 10 Top 10 categories of apps reviewed by the detected promotional attackers.

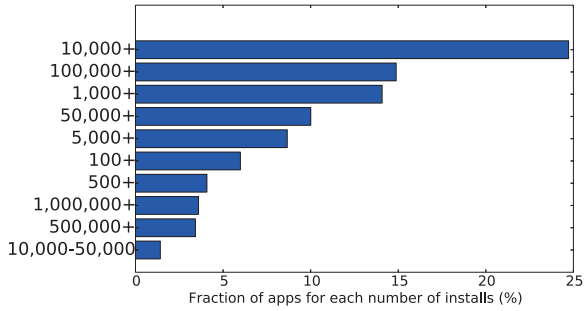

Fig. 11 Top 10 number of installs for apps reviewed by the detected promotional attackers.

We examine the top 241 detected promotional attackers who only reviewed malicious apps, and find that 72 of them reviewed malicious apps before these malicious apps were detected by VirusTotal. Among all the apps reviewed by these 72 promotional attackers, 217 apps were labeled as malicious app by VirusTotal. It is worth noting that other apps reviewed by the promotional attackers might also be suspicious.

In summary, PADetective discovered 289 K reviewers as potential promotional attackers. They posted reviews for 136 K apps, which included 21 K malicious apps. Among the top 1,000 reviewers who were flagged as promotional attackers with high confidence, 136 reviewers posted reviews only for malicious apps, and another 113 reviewers posted reviews for apps, most of which were detected as malicious apps. The result also suggests that PADetective could be used to detect malicious apps in the early stage of distribution.

5.2 Case Studies

Herein, we detail two PAs to demonstrate the effectiveness of our PADetective system.

promotional attackers in the wild. Table 8 lists a set of apps reviewed by a promotional attacker. This promotional attacker gave high ratings and posted similar positive reviews for seven malicious apps on the same day. These malicious apps belonged to different category, were not very popular, and were created by the same developer. Moreover, the average of the difference between the true reputation scores and ratings was larger than 1.0, indicating that the reviewer attempted to promote all the malicious apps using high ratings. This example illustrates the common features of promotional attackers in the wild.

Detecting previously unknown malicious apps. As shown in **Table 9**, this promotional attacker gave high ratings and wrote very short positive reviews for three malicious apps on the same day. Moreover, all the posting times are earlier than the first submission time in VirusTotal. The apps reviewed by this promotional attacker would be more likely to be malicious. The security expert and market operator can therefore discover new, previously unknown malicious apps by analyzing the apps related to this promotional attacker detected by PADetective.

6. Discussion

This section discusses some limitations of PADetective and future research directions.

Evasion. Advanced attackers may evade the PADetective system by employing lots of user accounts with different names and/or mimicking the reviewing behaviors of normal users. It is worth noting that such evasion strategies require much more resources and efforts. For example, attackers may acquire lots of fake user accounts and use each account to just post one comment in order to degrade the detection accuracy of PADetective. However, since mobile app stores (e.g., Google Play) usually adopt advanced techniques [20] to deter automated account registration, it will cost the attackers lots of resources and efforts to create many accounts and it does not benefit the attackers if these accounts are just used to post one comment. Note that the primary goal of the attackers is to increase the success rate of attacks with lower costs [21]. Even if an attacker can afford to adopt such an expensive approach, the stakeholders of mobile app stores can enhance PADetective with additional information about each account, such as IP address which could be correlated with user accounts to detect malicious users [22]. The attackers may also mimic the reviewing behaviors of normal users by writing short/long reviews, reviewing both legitimate and malicious apps, adjusting the posting time, and etc. It will also significantly increase the cost of attacks. We leave the challenge of differentiating such advanced attacks and human reviewers in future work.

Number of apps reviewed by each reviewer. PADetective does not consider reviewers who posted comments for only one or two apps. This constraint originates from the fact that computing some features such as entropy or coefficient variants requires more than two samples. In this work, we empirically set the number as 3 because increasing the number was not sensitive to the final outcomes. Since attackers usually employ the accounts to post a number of comments as we discussed above, we believe that this number is reasonable to capture promotional attackers. As the number of apps reviewed by a reviewer may exceed the threshold, 3, over time, PADetective could identify them by continuously collecting and analyzing the comments. We will construct a real-time detection system for fetching and examining UGC and the metadata continuously in future work.

7. Related Work

This section introduces mostly related work in three categories.

7.1 UGC analysis

Review Analysis. Kong et al. [23] designed AutoREB to automatically identify users' concerns on the security and privacy of mobile apps. They applied the relevance feedback technique for the semantic analysis of user reviews and then associated the results of the user review analysis to the apps' behaviors by using the crowd-sourcing technique. Mukherjee et al. [6], [7] proposed new approaches to detect fake reviewer groups from Amazon product reviews. They first used a frequent itemset mining method to identify a set of candidate groups, and then adopted several behavioral models based on the relationships among groups such as the review posting time and similarities. Fu et al. [24] proposed WisCom to provide important insights for end-users, developers, and potentially the entire mobile app ecosystem. They leveraged sentiment analysis, topic model analysis, and time-series analysis to examine over 13 M user reviews. Gomez et al. [25] analyzed user reviews and permissions using an unsupervised learning approach to detect apps that contain bugs and errors.

Rating Analysis. Xie et al. [3] proposed a new method for discovering colluded reviewers in app stores. They built a relation graph based on the ratings and the deviations of the ratings, and applied a graph cluster algorithm to detect collusion groups. Oh et al. [9] developed an algorithm that calculates the confidence score of each app. Market operators can replace the average rating of each app with the confidence score to defend against rating promotion/demotion attacks. Lim et al. [26] devised an approach to measure the degree of spam for each reviewer based on the rating behaviors, and evaluated them using an Amazon review dataset.

Previous works [3], [6], [7] are closely related to our work. The major differences between PADetective and Xie et al. [3] is the scalability. More precisely, their system is not scalable because it is not possible to build a tie graph of large-scale dataset in physical memory. Moreover, they performed the evaluation on a small and local dataset (200 apps collected from the china apple store). In contrast, since our detection model uses static features, our system can conduct large-scale analysis. Moreover, we investigate the prevalence of PAs in the official Android app store by collecting information on more than 1 M apps.

The method of review analysis is the main difference between PADetective and Refs. [6], [7]. Since they aimed to identify copy reviews used by spammers, their method only extracts the similar reviews in keyword level, e.g., "good app" and "good apps". Since users can express the same opinion using different words and expressions, e.g., "nice app" and "good app," we leveraged the state-of-the-art NLP technique called Paragraph vector [8] to extract similar reviews at the semantic level for better accuracy.

7.2 Metadata Analysis

Xie et al. [4] studied the mobile app reviews traded on the underground market. They analyzed the metadata of the promoted apps collected from the underground market, including average ratings, total number of reviewers, category distributions, and developers. WHYPER [27] was the first system that analyzes text descriptions semantically to perform risk assessments of mobile apps. Qu et al. [28] developed AutoCog for measuring description-to-permission fidelity in Android applications. Peng et al. [29] designed an app risk scoring and ranking system based on probabilistic generative models in order to improve risk communication mechanism for Android apps. The system was trained on the metadata including the developer name, the category, and the set of permissions requested by the app. Yu et al. [30] proposed a novel approach to automate the detection of incomplete, incorrect and inconsistent privacy policy by combining description and code analysis. Unlike these systems, PADetective can extract features from not only the metadata mentioned above but

also UGC in a more comprehensive way.

7.3 App Analysis

Several studies [31], [32], [33], [34], [35], [36], [37], [38], [39], [40] have analyzed the permission, API call, and runtime behavior derived from code to detect malicious apps or prevent system and application vulnerabilities from being abused by attacker.

Kirin [31] is a lightweight system that can flag potential malware applications at the time of installation using a set of security rules that match malware characteristics. DroidScope [32] rebuilds both the OS-level and Java-level semantics simultaneously and seamlessly to unveil malicious intent and the inner workings of a malware application quickly. DroidRanger [33] applies a heuristics-based filtering scheme to discover unknown malicious apps from real-world datasets. RiskRanker [34] can automatically identify zero-day Android malware by examining apps' runtime behaviors. Unlike DroidRanger, it does not require malware specimens to detect zero-day malware. DREBIN [35] is a lightweight and automatic Android malware detection system. DroidMiner [37] can automatically mine malicious program logic from known Android malware using behavioral graph and machine-learning techniques. PADetective complements to these studies by investigating the relationships among UGC, metadata, and malicious apps to detect promotional attackers and reveals malicious apps.

8. Conclusion

In this study, we propose and develop PADetective, which can identify unknown promotional attackers in mobile app stores, using UGC and metadata as well as machine-learning techniques. We extracted 15 features from the UGC and metadata and selected the most suitable machine-learning methods for our detection model. The extensive experiment results demonstrate that our detection scheme can achieve a high true positive rate of up to 90% and a low false positive rate (i.e., 5.8%). We also applied PADetective to a large-scale analysis of unlabeled reviewer data; it detected 289 K reviewers as potential promotional attackers, who posted reviews to 136 K apps, including 21 K malicious apps. Moreover, the large-scale evaluation and case study analysis illustrate that PADetective can effectively and efficiently discover previous unknown malicious apps.

Acknowledgments A part of this work was supported by JSPS Grant-in-Aid for Scientific Research (KAKENHI) B, Grant number JP16H02832. A part of this work was also supported by a Grant for Non-Japanese Researchers from the NEC C&C Foundation and a Waseda University Grant for Special Research Projects (Project number: 2016S-055).

References

[1] Statista Inc.: Number of apps available in leading app stores as of June 2016, available from ⟨http://goo.gl/JnBkmY⟩.
[2] Ganguly, R.: App Store Optimization - A Crucial Piece of the Mobile App Marketing Puzzle (2013), available from ⟨https://blog.kissmetrics.com/app-store-optimization/⟩.
[3] Xie, Z. and Zhu, S.: GroupTie: toward hidden collusion group discovery in app stores, *Proc. ACM WiSec* (2014).
[4] Xie, Z. and Zhu, S.: AppWatcher: Unveiling the underground market of trading mobile app reviews, *Proc. ACM WiSec* (2015).
[5] The FTC′s Endorsement Guides: What People Are Asking, available from ⟨http://goo.gl/3875GT⟩ (2015).
[6] Mukherjee, A., Liu, B., Wang, J., Glance, N.S. and Jindal, N.: Detecting group review spam, *Proc. WWW* (2011).
[7] Mukherjee, A., Liu, B. and Glance, N.S.: Spotting fake reviewer groups in consumer reviews, *Proc. WWW* (2012).
[8] Le, Q.V. and Mikolov, T.: Distributed Representations of Sentences and Documents, *Proc. ICML* (2014).
[9] Oh, H., Kim, S., Park, S. and Zhou, M.: Can You Trust Online Ratings? A Mutual Reinforcement Model for Trustworthy Online Rating Systems, *IEEE Trans. Systems, Man, and Cybernetics: Systems*, Vol.45, No.12 (2015).
[10] Viennot, N., Garcia, E. and Nieh, J.: A measurement study of google play, *Proc. ACM SIGMETRICS* (2014).
[11] Google Play reviews collection service, available from ⟨https://play.google.com/store/getreviews⟩.
[12] Natural Language Toolkit, available from ⟨http://www.nltk.org⟩.
[13] TextBlob: Simplified Text Processing, available from ⟨http://textblob.readthedocs.io/en/dev/⟩.
[14] gensim:topic modelling for humans, available from ⟨https://radimrehurek.com/gensim/⟩.
[15] Feature Selection, available from ⟨http://scikit-learn.org/stable/modules/feature_selection.html⟩.
[16] Developer Policy Center, available from ⟨http://goo.gl/yA0qUb⟩.
[17] scikit-learn:machine learning in Python, available from ⟨http://scikit-learn.org/stable/⟩.
[18] VirusTotal- Free Online Virus, Malware and URL Scanner, available from ⟨https://www.virustotal.com⟩.
[19] Breiman, L.: Random Forests, *Machine Learning*, Vol.45, No.1, pp.5–32 (online), DOI: 10.1023/A:1010933404324 (2001).
[20] El Ahmad, A.S., Yan, J. and Ng, W.-Y.: CAPTCHA Design: Color, Usability, and Security, *IEEE Internet Computing*, Vol.16, No.2 (2012).
[21] Liu, B., Nath, S., Govindan, R. and Liu, J.: DECAF: Detecting and Characterizing Ad Fraud in Mobile Apps, *Proc. NSDI* (2014).
[22] Zhao, Y., Xie, Y., Yu, F., Ke, Q., Yu, Y., Chen, Y. and Gillum, E.: BotGraph: Large Scale Spamming Botnet Detection, *Proc. NSDI* (2009).
[23] Kong, D., Cen, L. and Jin, H.: AUTOREB: Automatically Understanding the Review-to-Behavior Fidelity in Android Applications, *Proc. ACM CCS* (2015).
[24] Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J.I. and Sadeh, N.M.: Why people hate your app: making sense of user feedback in a mobile app store, *Proc. ACM KDD* (2013).
[25] Gomez, M., Rouvoy, R., Monperrus, M. and Seinturier, L.: A Recommender System of Buggy App Checkers for App Store Moderators, *Proc. ACM MOBILESoft* (2015).
[26] Lim, E., Nguyen, V., Jindal, N., Liu, B. and Lauw, H.W.: Detecting product review spammers using rating behaviors, *Proc. ACM CIKM* (2010).
[27] Pandita, R., Xiao, X., Yang, W., Enck, W. and Xie, T.: WHYPER: Towards Automating Risk Assessment of Mobile Applications, *Proc. USENIX Sec.* (2013).
[28] Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T. and Chen, Z.: AutoCog: Measuring the Description-to-permission Fidelity in Android Applications, *Proc. ACM CCS* (2014).
[29] Peng, H., Gates, C.S., Sarma, B.P., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C. and Molloy, I.: Using probabilistic generative models for ranking risks of Android apps, *Proc. ACM CCS* (2012).
[30] Yu, L., Luo, X., Liu, X. and Zhang, T.: Can We Trust the Privacy Policies of Android Apps?, *Proc. DSN* (2016).
[31] Enck, W., Ongtang, M. and McDaniel, P.D.: On lightweight mobile phone application certification, *Proc. ACM CCS* (2009).
[32] Yan, L. and Yin, H.: DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis, *Proc. USENIX Sec.* (2012).
[33] Zhou, Y., Wang, Z., Zhou, W. and Jiang, X.: Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets, *Proc. NDSS* (2012).
[34] Grace, M.C., Zhou, Y., Zhang, Q., Zou, S. and Jiang, X.: RiskRanker: scalable and accurate zero-day android malware detection, *Proc. ACM MobiSys* (2012).
[35] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H. and Rieck, K.: DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket, *Proc. NDSS* (2014).
[36] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A. and Shastry, B.: Towards Taming Privilege-Escalation Attacks on Android, *Proc. NDSS* (2012).
[37] Yang, C., Xu, Z., Gu, G., Yegneswaran, V. and Porras, P.A.: DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications, *Proc. ESORICS* (2014).

[38] Chen, J., Chen, H., Bauman, E., Lin, Z., Zang, B. and Guan, H.: You Shouldn't Collect My Secrets: Thwarting Sensitive Keystroke Leakage in Mobile IME Apps, *Proc. USENIX Sec.* (2015).
[39] Lee, B., Lu, L., Wang, T., Kim, T. and Lee, W.: From Zygote to Morula: Fortifying Weakened ASLR on Android, *Proc. IEEE Symposium on Security and Privacy* (2014).
[40] Wei, F., Roy, S., Ou, X. and Robby: Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps, *Proc. ACM CCS* (2014).

Editor's Recommendation

This paper establishes a new problem, namely promotional attacks, and proposes a novel machine learning-based method to identify unknown promotional attackers. The paper also reports evaluation results using large real datasets along with insightful discussions, which reliably shows the effectiveness of the proposed method, and thus is selected as a recommended paper. (Masayuki Terada, program chair of Computer Security Symposium 2016 (CSS2016))

Bo Sun was born in 1984. He received his B.E degree in computer science from Jilin University in 2007, and M.E degree in Information and Media from Yokohama National University in 2012. He is currently a research associate in the Department of Computer Science and Engineering, Waseda University. His research interest is network security and mobile security.

Xiapu Luo received his Ph.D. degree in computer science from The Hong Kong Polytechnic University. He was a postdoctoral research fellow with the Georgia Institute of Technology. He is currently an assistant professor with the Department of Computing, The Hong Kong Polytechnic University. His current research focuses on smartphone security and privacy, network security and privacy, and Internet measurement.

Mitsuaki Akiyama received his M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Japan in 2007 and 2013. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2007, he has been engaged in research and development on network security, especially honeypot and malware analysis. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.

Takuya Watanabe recieved his M.E. degree in computer science and engineering from Waseda University, Japan in 2016. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2016, he has been engaged in research and development of mobile/IoT security. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.

Tatsuya Mori is currently an associate professor at Waseda University, Tokyo, Japan. He received his B.E. and M.E. degrees in applied physics, and Ph.D. degree in information science from the Waseda University, in 1997, 1999 and 2005, respectively. He joined NTT lab in 1999. Since then, he has been engaged in the research of measurement and analysis of networks and cyber security. From Mar 2007 to Mar 2008, he was a visiting researcher at the University of Wisconsin-Madison. He received Telecom System Technology Award from TAF in 2010 and Best Paper Awards from IEICE and IEEE/ACM COMSNETS in 2009 and 2010, respectively. Dr. Mori is a member of ACM, IEEE, IEICE, IPSJ, and USENIX.