Characterizing Promotional Attacks in Mobile App Store

Bo Sun[1](✉), Xiapu Luo[2], Mitsuaki Akiyama[3], Takuya Watanabe[3], and Tatsuya Mori[1]

[1] Department of Computer Science and Communications Engineering, Waseda University, Shinjuku, Japan
{sunshine,mori}@nsl.cs.waseda.ac.jp

[2] Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
csxluo@comp.polyu.edu.hk

[3] NTT Secure Platform Laboratories, NTT Corporation, Tokyo, Japan
akiyama.mitsuaki@lab.ntt.co.jp, watanabe@nsl.cs.waseda.ac.jp

Abstract. Mobile app stores, such as Google Play, play a vital role in the ecosystem of mobile apps. When users look for an app of interest, they can acquire useful data from the app store to facilitate their decision on installing the app or not. 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. Unfortunately, *miscreants* also exploit such channels to conduct *promotional attacks* (PAs) that lure victims to install malicious apps. In this paper, we propose and develop a new system called *PADetective* to detect miscreants who are likely to be conducting promotional attacks. Using a dataset with 1,723 of labeled samples, we demonstrate that the true positive rate of detection model is 90%, with a false positive rate of 5.8%. We then applied *PADetective* to a large dataset for characterizing the prevalence of PAs in the wild and find 289 K potential PA attackers who posted reviews to 21 K malicious apps.

Keywords: Mobile app store · Promotional attacks · Machine learning

1 Introduction

With more than four million apps [20], 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 various 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 [12], recent studies report that some developers adopt unfair approaches to manipulate their apps' ratings and reviews [22,23], even if such behaviors are

© Springer Nature Singapore Pte Ltd. 2017

prohibited by FTC [9] and app markets. Note that attackers also employ such 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 [22] and colluded reviewers [23], 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 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 [22], and requiring the similar reviews in keyword level affects the accuracy of [17,18]. 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 the state-of-the-art NLP technique (i.e., *Paragraph vector* [14]) 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 [19] 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% (Sect. 4).
- 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 (Sect. 5).
- 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 (Sect. 5).

2 Problem Statement

This section specifies the problem we address in this paper by first presenting the high-level overview of the problem and then describing its mathematical formulation. Figure 1 presents the high-level overview of the problem. Although this work targets Google Play, the model is applicable to other mobile app stores 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.

Our goal is to determine whether a given reviewer is a PA attacker or not by analyzing the UGC and the metadata associated with apps commented on or rated by the reviewer. To achieve it, we first extract a feature vector from the UGC and app metadata, and then train a classifier using labeled data. After that, we apply the trained classifier to differentiate legitimate reviewers and a PA attackers.

To formulate the problem in a mathematical way, we introduce the variables summarized in Table 1. It is worth noting 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 Sect. 6. Of the variables 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 $\mathbf{F}(r_i) = \{f_1^i, f_2^i, \dots f_{15}^i\}$ 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.

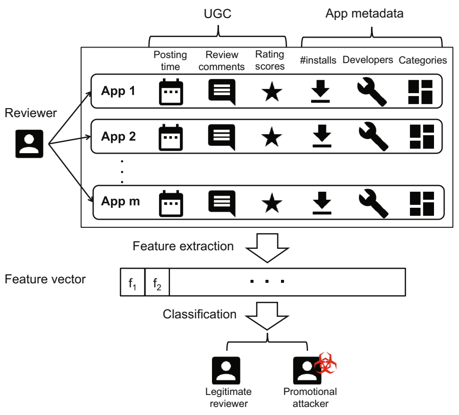

Fig. 1. High-level overview of the problem.

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 which 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. $j = 1, 2, \ldots, m_i$

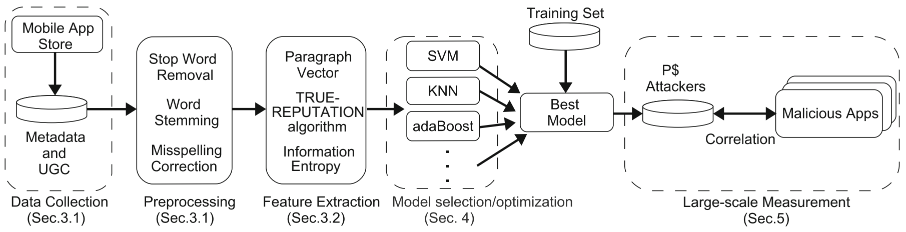

Fig. 2. Overview of PADetective.

3 PADetective System

This section details PADetective (Fig. 2), especially its four major components including: data collection, data preprocessing, feature extraction, and detection.

3.1 Data Collection and Preprocessing

Collection. We first create a list of apps to be downloaded by using the list of package names in [21]. Then, we collect metadata for each app by accessing its description page according to its package name and employing our HTML parser to extract the metadata in the page. Moreover, we develop a UGC crawler by leveraging the review collection API [4] provided by Google Play Store. Figure 3 shows the statistics of the number of reviews in each app. Note that the Google

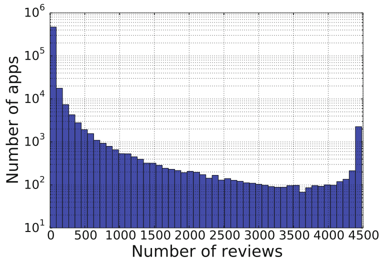

Fig. 3. Histogram for the number of reviews in each app.

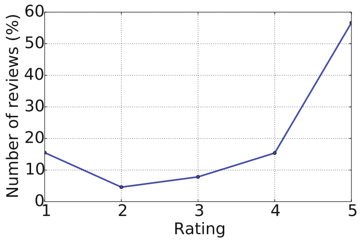

Fig. 4. Percentage of review numbers with different rating.

Play review collection service only allows 4,500 most recent reviews to be crawled for each app. To circumvent this limitation, we could fetch the reviews continuously 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. 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.

Preprocessing. Before creating the feature vector for the classifier, we develop a 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. **Step 2:** Extract the reviewers who have commented on at least three apps. The limitation introduced by this step is discussed in Sect. 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 misspelled English words for all the reviews. For Steps 3–8, we implement the natural language processing based on NLTK [5] and TextBlob [7]. TextBlob enables us to realize language detection and spelling correction. After data preprocessing, our dataset for feature extraction includes 2,606,791 reviewers.

3.2 Feature Extraction

We profile each reviewer r_i using 15 features extracted from UGC and metadata. These features form a feature vector $\mathbf{F}(r_i) = \{f_1^i, f_2^i, \dots f_{15}^i\}$, and are described as follows.

f_1^i: **Day intervals.** PA Attackers are likely to launch PA attacks within a short day interval. For example, Xie and Zhu found that reviewers hired by app promotion web services tend to complete their review promotion missions within 120 days [23]. 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.** PA Attackers are likely to write reviews within the same day, because they may use automated posting process or want to finish the task as quickly as possible. To measure the proportion of same-day reviews, we defined f_2^i using the information entropy: $f_2^i = H(X) = -\sum_{j=1}^{m_i} P(t_{ij}) \log P(t_{ij})$, where $P(t_{ij})$ is the frequency of same-day reviews: t_{ij}/sum and $sum = \sum_{j=1}^{m_i} t_{ij}$ is the sum of days reviewed by reviewer r_i. If all the reviews are posted on the same day, the entropy of the post time will be 0.
f_3^i: **Bi-gram matching.** PA 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, $f_3^i = \sum_{j=1}^{m_i} \sum_{k=1}^{m_i} cosim(c_{ij}, c_{ik})/m_i^2$. 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 the Paragraph Vector (PV) algorithm [14], because it performs a semantic analysis in discovering similar words and expressions. By applying the PV algorithm realized in the Python library gensim [3] to $57,868,301$ reviews in our dataset, we get the predicted model after around 1 h. We defined f_4^i as the average of the similarity scores predicted from the trained model for each pair of reviews. $f_4^i = \sum_{j=1}^{m_i} \sum_{k=1}^{m_i} D(c_{ij}, c_{ik})/m_i^2$, Where D is the distance of two different documents computed by PV algorithm. Table 2 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.** PA attackers usually post positive reviews to promote apps for monetary benefit and/or luring victims to install malicious apps. Sentiment analysis classifies the attitude of a text into three categories: negative, neutral, positive. Using sentiment analysis, we could reveal potential PA attackers if all the reviews are positive. We use TextBlob [7] 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. We define f_5^i as the average score for each

Table 2. 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 3. 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

pair of reviews predicted by the sentiment analysis classifier. Table 3 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. It shows that our classifier can correctly identify the sentiment of the reviews.

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.

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, PA 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 [19], which takes into account the user confidence in terms of user activity, user objectivity, and user consistency. f_7^i is computed as: $f_7^i = \sum_{i=1}^{m_i}(s_{ij} - u_{aj})/m_i$, where m_i is the number of apps reviewed by reviewer r_i. a is an app and u_a is true reputation score for app a.

f_8^i: **Average ratings.** Since PA attackers 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. It is the ratio of the standard deviation to the mean: $f_9^i = \sigma(\mathbf{S}_i)/\sum_{j=1}^{m_i} s_{ij}$, where σ is standard deviation and $\mathbf{S}_i = \{s_{i1}, \ldots, s_{im_i}\}$. If a reviewer posts identical ratings, f_9^i will be 0.

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. $f_{10}^i = \sum_{j=1}^{m_i} n_{ij}/m_i$.

f^i_{11}: **Coefficient of variation of the number of installs.** To measure the distribution of the number of installs, we define f^i_{11} as the coefficient of variation of the number of installs for reviewer r_i. The computation of f^i_{11} can be referred to the equation defined by f^i_{10}. If a reviewer posts reviews to apps with the same number of installs, the coefficient of variation will be 0.

f^i_{12}: **Developer Entropy.** PA attackers are more likely to promote apps from the same developer because the targeted malicious apps should be associated with each other. Therefore, we defined f^i_{12} as the entropy of developer for reviewer r_i. The computation of f^i_{12} can be referred to the equation defined by f^i_2. If a reviewer only posts reviews for apps from the same developer, his/her f_{12} will be 0.

f^i_{13}: **Category Entropy.** PA attackers tend to promote apps having a small number of distinct categories, possibly due to the automated posting process. Similar to f^i_{12}, we defined f^i_{13} as the entropy of category for reviewer r_i. The computation of f^i_{13} can also be referred to the equation defined by f^i_2. If a reviewer only posts reviews for apps having a small number of distinct categories, his/her f_{13} will be 0.

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

f^i_{15}: **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^i_{15} as the number of digits and symbols in the reviewer names.

3.3 Effectiveness of Feature and Description of Detection Model

Effectiveness of feature. To demonstrate how our features facilitate the detection, we compute the importance of our features. For the space limitation, we present the top-3 features that had the largest contributions (f^i_1: Day intervals, f^i_{10}: Average number of installs, f^i_{12}: Developer Entropy). We extracted these three features by using tree-based feature selection method [2], which uses forests of trees to evaluate the importance of features.

Figure 5 shows the 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 [1]. 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.

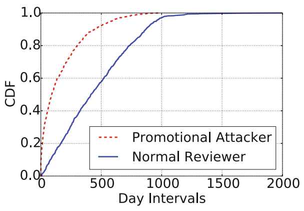

Fig. 5. f_1^i: Day intervals.

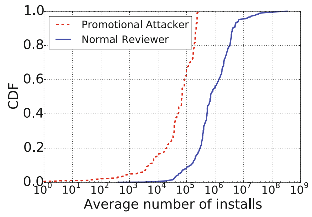

Fig. 6. f_{10}^i: Average number of installs.

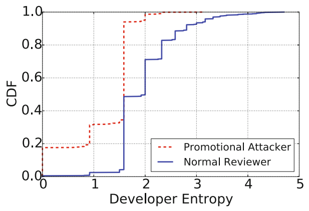

Fig. 7. f_{12}^i: Developer Entropy

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.

Description of detection model. We build our detection model using the library scikit-learn [6] because it is efficient, and implement several supervised learning algorithms, including support vector machine (SVM), k-nearest neighbor (KNN), random forest, decision tree, and adaBoost. To determine the best algorithm and parameters, we test the algorithms and parameters using our labeled dataset. The detailed model selection process and its results are presented in Sect. 4. Finally, we use 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.

Training Dataset. We first generate the training dataset with the ground truth. Since legitimate reviewers may comment bad apps and/or post reviews to malicious apps, we define a PA attacker as a reviewer who only posts reviews to malicious apps and comments at least three malicious apps. We determine whether an app was malicious by submitting the app to VirusTotal [8] 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. We also note that VirusTotal usually classifies malicious apps into two categories: malware and adware. 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.

Evaluation Method. 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 used in those algorithms (e.g., for random forest, we set parameter "n_estimators" to a set of values: 50, 100, 150, 200, 250). Then, we evaluate the machine learning algorithms with different parameters through 10-fold cross-validation. Finally, we select the 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. 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 $FPR = FP/(FP+TN)$, $FNR = FN/(TP+FN)$, and $ACC = (TP+TN)/(TP+TN+FP+FN)$, 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.

Table 4. Classification accuracy. The means and standard deviations are calculated using 10-times 10-fold cross-validation tests for each machine learning algorithm.

Machine learning Algorithm	ACC		FPR		FNR	
	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

Evaluation Result. Table 4 lists the accuracy of different machine learning algorithms used by PADetective. Most of these algorithms discover the PA attackers with high accuracy and low false negative or false positive rate. Among the five machine learning algorithms we tested, RandomForest achieves the highest accuracy (i.e., 0.933) 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.

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 PA attackers who had posted reviews for a

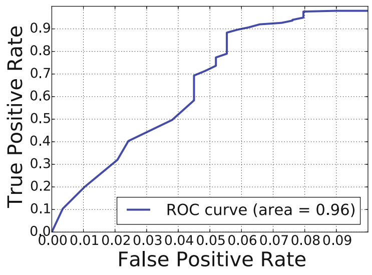

Fig. 8. Evaluation of detection model using test set.

period of two years or longer. On the other hand, PADetective wrongly flagged the legitimate reviewers whose behaviors were similar to a PA attacker (e.g., their reviews seemed to be fake, but the apps were not flagged as malware/adware by VirusTotal). Note 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 8 shows that it can achieve 90% true positive rate with low false positive rate of 5.8%.

5 Promtional Attacks in the Wild

Using PADetective, we examined a large-scale data collected from the Google Play Store, and found 289,000 potential PA attackers from 2,605,068 reviewers. Table 5 summarizes the number of reviewers/apps detected by PADetective. The number of unique malicious apps reviewed by the potential PA attackers was 20,906, accounting for approximately 65% of the malicious apps reviewed by all observed reviewers. Many malicious apps having reviews were associated with the potential PA attackers. Moreover, the majority of malicious apps detected by VirusTotal had no user reviews. It may be due to the fact that the malicious apps were detected and deleted by mobile app stores in the early stage of distribution, and hence there are no comments on such apps. Another possibility is that mobile app stores deleted both malicious apps and their information including reviews simultaneously, and therefore we can not collect the reviews. We ranked

Table 5. 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 probability	$1,000$	$2,904$	486	148

the reviewers in descending order according to the probability of being a PA attacker, and investigated top 1,000 reviewers. 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 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 acknowledge that using the online virus checkers might lead to false detection, and leave the checking of those undetected apps in future work.

Figure 9 shows the top 10 categories of the apps reviewed by PA attackers. Three categories (approximately 15% in total) are related to games, which was the primary target of the PAs. To study the impact of apps promoted by PA attackers, Fig. 10 illustrates the top 10 number of installs of the apps reviewed by PA attackers. It shows 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 PA attackers can be used to discover malicious apps. More precisely, we compare the time when the PA 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 malicious apps that have not been listed in Virustotal. We examine the top 241 detected PA 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 PA attackers might also be suspicious.

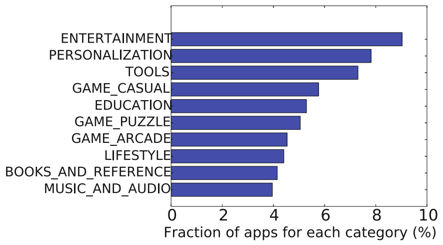

Fig. 9. Top 10 categories of apps reviewed by the detected promotional attackers.

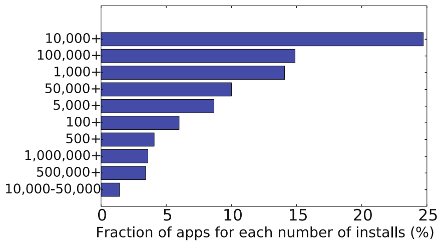

Fig. 10. Top 10 number of installs for apps reviewed by the detected promotional attackers.

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 [10] 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 [16]. Even if an attacker affords 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 [24]. 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 require 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

Review Analysis. Kong et al. [13] 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. [17,18] 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. [11] 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.

Rating Analysis. Xie et al. [22] 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. [19] 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. [15] 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.

Among previous works mentioned above, [17,18,22] are closely related to our work. The major differences between PADetective and Xie et al. [22] 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 [17,18]. 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 [14] to extract similar reviews at the semantic level for better accuracy.

8 Conclusion

In this study, we developed PADetective to detect PA attackers in mobile app stores using UGC and metadata as well as machine-learning techniques. The large-scale evaluation revealed that we can exploit the PA attackers identified by PADetective to discover potentially malicious apps effectively and efficiently. 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.

Acknowledgements. 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. Developer policy center. http://goo.gl/yA0qUb
2. Feature selection. http://scikit-learn.org/stable/modules/feature_selection.html
3. gensim:topic modelling for humans. https://radimrehurek.com/gensim/
4. Google play reviews collection service. https://play.google.com/store/getreviews
5. Natural language toolkit. http://www.nltk.org
6. scikit-learn:machine learning in python. http://scikit-learn.org/stable/
7. Textblob: Simplified text processing. http://textblob.readthedocs.io/en/dev/
8. Virustotal- free online virus, malware and url scanner. https://www.virustotal.com
9. The FTC's endorsement guides: What people are asking (2015). http://goo.gl/3875GT
10. El Ahmad, A.S., Yan, J., Ng, W.-Y.: Captcha design: color, usability, and security. IEEE Internet Comput. **16**(2), 44–51 (2012)
11. Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J.I., Sadeh, N.M.: Why people hate your app: making sense of user feedback in a mobile app store. In: Proceedings of the ACM KDD (2013)
12. Ganguly, R.: App. store optimization - a crucial piece of the mobile app marketing puzzle (2013). https://blog.kissmetrics.com/app-store-optimization/
13. Kong, D., Cen, L., Jin, H.: AUTOREB: automatically understanding the review-to-behavior fidelity in android applications. In: Proceedings of the ACM CCS (2015)
14. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the ICML (2014)
15. Lim, E., Nguyen, V., Jindal, N., Liu, B., Lauw, H.W.: Detecting product review spammers using rating behaviors. In: Proceedings of the ACM CIKM (2010)
16. Liu, B., Nath, S., Govindan, R., Liu, J.: DECAF: detecting and characterizing ad fraud in mobile apps. In: Proceedings of the NSDI (2014)
17. Mukherjee, A., Liu, B., Glance, N.S.: Spotting fake reviewer groups in consumer reviews. In: Proceedings of the WWW (2012)
18. Mukherjee, A., Liu, B., Wang, J., Glance, N.S., Jindal, N.: Detecting group review spam. In: Proceedings of the WWW (2011)
19. Oh, H., Kim, S., Park, S., Zhou, M.: Can you trust online ratings? A mutual reinforcement model for trustworthy online rating systems. IEEE Trans. Syst. Man Cybern. Syst. **45**(12), 1564–1576 (2015)
20. Statista Inc.: Number of apps available in leading app stores as of June 2016. http://goo.gl/JnBkmY
21. Viennot, N., Garcia, E., Nieh, J.: A measurement study of google play. In: Proceedings of the ACM SIGMETRICS (2014)
22. Xie, Z., Zhu, S.: Grouptie: toward hidden collusion group discovery in app stores. In: Proceedings of the ACM WiSec (2014)
23. Xie, Z., Zhu, S.: Appwatcher: unveiling the underground market of trading mobile app reviews. In: Proceedings of the ACM WiSec (2015)
24. Zhao, Y., Xie, Y., Yu, F., Ke, Q., Yu, Y., Chen, Y., Gillum, E.: Botgraph: large scale spamming botnet detection. In: Proceedings of the NSDI (2009)