Towards Automatically Detecting Promotional Attacks in Mobile App Store

Bo Sun[1] Mitsuaki Akiyama[2] Tatsuya Mori[1]

Abstract: Mobile app store is an important component of mobile ecosystem such as the Google Play Store. It is convenient for mobile users to gain information such as ratings/reviews (called *user-generated content* (UGC)) and the number of app downloads, category (called Metadata) from mobile app store. The popularity of mobile app store is leading to a rapidly growing number of security and privacy problems. i.e., a miscreant may promote many malicious apps with fake ratings/reviews in a very short period of time in order to attract more victims to download/install a malicious app. To address these issues, we developed a system called *PADetective* that identifys miscreants who are likely to be performing promotional attacks. We collected a large-scale dataset of approximately 20 M unique users, 1 M apps and 57 M reviews from real world and demonstrated that PADetective system is able to achieve a high detection accuracy as 95.4% with a false-positive rate of 8%. We then conducted a large-scale analysis based on real world dataset and found that our system can be applied to detect malware that has not been discovered by anti-virus checkers.

Keywords: Mobile app store, Promotional attack, Machine learning

1. Introduction

One of the key success factors of the widespread of smartphone today is attributed to the core of its ecosystem – mobile app store. Mobile app stores, such as Google Play or Apple App Store, play a vital role in distributing apps to end users in an efficient and usable way. The mobile app stores provide various information about apps; i.e., descriptions, preview screenshots, number of installs, names of developers, categories of apps, etc. In addition, many of mobile app stores are composed of *reputation system*, i.e., user can post their ratings or reviews of apps. These information is useful for an user to determine whether he/she wants to install an app.

Reputation systems generally have been widely studied in various domains. To ensure the reliability of a reputation system, it is essential to cope with various noises imposed to the system. Such noises include spam and *fake opinion*, which we focus on in this work. The motivation to post a fake opinion is that by posting such fake opinions a lot, one can take control of the reputation for an item. For instance, by expressing positive/negative opinions about an item, one can promote/demote it. Several works have attempted to detect fake opinions in various domains [1], [2], [3]. Xie et al. [4], [5] reported that mobile app stores are also suffering from the same problem as the domains mentioned above.

Given the nature of reputation systems, we employ the following threat model, which we call the *promotional attack* (PA) threat model. First, a miscreant publishes a malicious or a potentially harmful app on an app store. The miscreant then posts several good ratings or reviews using different IDs to promote the app. After the app is promoted, several users may install it without realizing that it is a bad app. This leads to our first research question:

Is the threat of PA in the mobile app store real?

To answer this research question, this study employed a large-scale measurement study using 1 M apps collected from the official Android app store. We first collected 57 M reviews posted by 14 M users for all the apps. We then identified and extracted malicious apps using multiple anti-virus checkers. Finally, we extracted users who posted reviews or ratings to at least three apps classified as malicious. This procedure revealed that at least 723 users were associated with PA for 10,845 malicious apps in our dataset, verifying that the security threat of PA in the mobile app stores is *real*. The next question we wanted to ask was:

Can we characterize PA in the wild?

Answering the above question was not straightforward for two reasons. First, because anti-virus checkers have imperfect detection rates, this study may miss some malicious apps. Second, some users may post reviews to not only malicious apps but also benign apps as a means of evasion. To address this issue, we developed a system called *PADetective*, which learns the features of *known promotional attackers* and then automatically detects *unknown promotional attackers* based on the learned features. To address "promotion", we need to identify *false reputation* by measuring the trustworthiness of user ratings. To this end, we leveraged the TRUE-REPUTATION algorithm [1], which is an algorithm to adjust a reputation based on the confidence of

[1] Waseda University
[2] NTT Secure Platform Laboratories

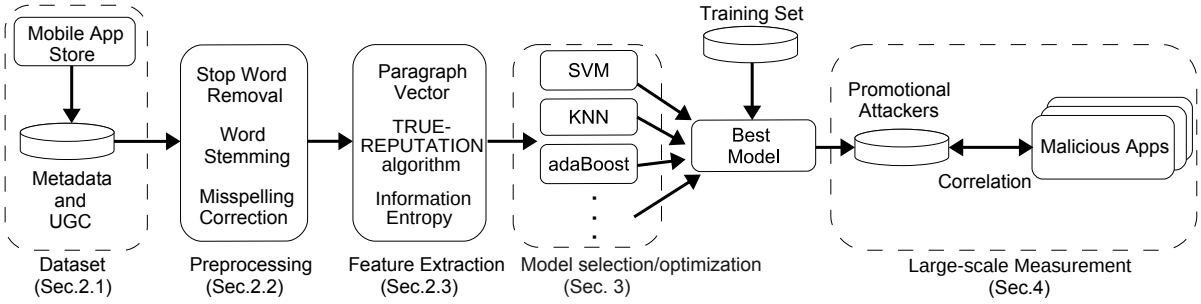

Fig. 1 Overview of the PADetective System.

customer ratings. Another challenge we addressed was to quantify the similarity of user reviews. As users can express the same opinion using different words and expressions, e.g., "nice app"and "good app", we need to measure the similarity of reviews in semantic level. To address this issue, we leveraged a state-of-the-art NLP technique called *Paragraph vector* [6], which works on top of the deep learning. *Paragraph vector* is an unsupervised algorithm that enables us to extract similar reviews in semantic level.

We tested the detection accuracy of the system using the 723 promotional attackers we extracted manually and found that the detection accuracy was 95.4% with a false positive rate of 8%. We then applied *PADetective* to an *unlabeled* set of 57 M reviews written by 20 M users for 1 M apps to characterize the prevalence of threats in the wild. We found that of the 2.6 M of reviewers, 289 K of them were flagged as potential promotion attackers. These reviewers posted reviews to 136 K of apps, which included 21 K of malicious apps. Among the top-1K 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 where more than half of apps were detected as malicious. Our analysis also found that PADetective can contribute to the detection of previously unknown malicious apps.

The chief contributions of this work can be summarized as follows.

- To characterize PA in the wild, we developed a system called *PADetective* that learns the features of known promotional attackers and automatically detects new promotional attackers with an accuracy of 95.4%.
- We applied several state-of-the-art techniques that can extract meaningful information from complex and subtle UGC and metadata.
- Through an extensive analysis of 57 M reviews posted by 15 M users for 1 M apps, we verified that the security threat of PA is real.
- We found that the *PADetective* system can be used to identify potentially malicious apps before they were submitted to public online virus checkers.

To the best of our knowledge, this is the first measurement study of promotional attackers in mobile app store. As our system can be used to discover malicious apps before they are detected by popular online anti-virus checkers, it can serve as an effective assistive tool for market operators and malware analysts. We believe that our approach sheds a new light on the analysis of UGC and metadata of app stores as a complementary channel to find malicious apps.

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

2. PADetective system

This section presents the architecture of the PADetective system. We describe in detail its four core components: data collection, data preprocessing, feature extraction, and detection.

2.1 Dataset

Data collection We describe how we gathered large-scare dataset from real world and what kind of statistics we collect from the dataset. We obtained UGC and metadata for approximately a million of apps from the official app store in November 2015. We first created a list of apps to be crawled. For this, we made use of the list of package names provided with PlayDrone [7]. Next, we collected metadata for each app. Collecting metadata is straightforward because they are static data. We accessed to the app description page by searching package name in the google play and employs our HTML parser to extract all the metadata presented in page. We then collected UGC for each app. Collecting UGC needs to address dynamic data, i.e., UGC is provided with the asynchronous communication interface. That means we need to communicate with server in order to gain each page of UGC. We developed a UGC crawler using the review collection API [8] provided by Google Play Store. If we send a HTTP request configuring package name and the number of page as parameter, we will receive a JSON file from HTTP response. As an effort to collect UGCs for a large number of apps, while following the acceptable use policy of the API, we deployed our crawler on 100 of servers each assigned different Global IP addresses. We note that The maximum number of reviews for each app was 4,500, because the Google Play review collection service only allows the 4,500 most recent reviews to be crawled for any app. The implications of this limitation will be discussed in Section 5.

Table 1 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	The number of download	The count of app downloaded by Mobile user, i.e. 1,000-5,000, 10,000+
	Category	The cluster name of apps with similar function, i.e. Entertainment, communication,Sports
	Creator	The name of an individual or a company who create the app

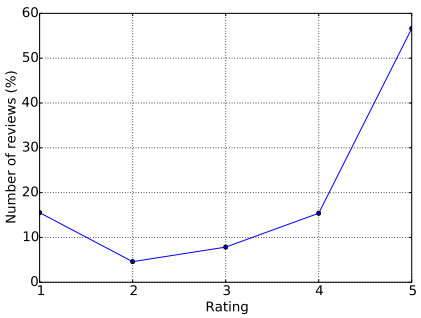

Fig. 2 Percentage of review numbers with different rating

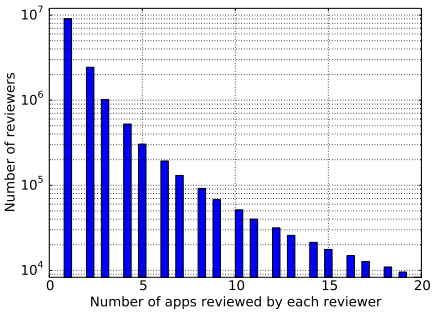

Fig. 4 Histogram for the number of apps reviewed by each reviewer

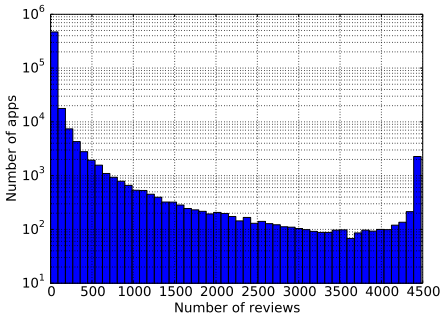

Fig. 3 Histogram for the number of reviews in each app

Data Statistics The statistics for the UGC and metadata we collected are presented in Table 1. We utilized all this information for feature extraction, the details of which will be presented in Section 2.3. We crawled a large-scale dataset of 20,211,517 unique users, 1,058,259 apps, and 57,868,301 reviews. Figure 2 presents the statistics for our collected rating data. The rating scale used in the Google Play Store ranges from 1 to 5. Over 55% of ratings are 5 stars, indicating that the reviewer has had an outstanding experience with the app. The tendency of giving high ratings hinders the detection of a PA behavior. Therefore, it is a challenging task to distinguish promotional attacker behavior from legitimate reviewers who are actually satisfied with an app. Figure 3 shows the statistics of the number of apps reviewed per reviewer. As shown in the figure, most reviewers tend to evaluate less than 3 apps due to the offline data collection method we used. The number of apps reviewed per reviewer will probably exceed 2 over time.

2.2 Data Preprocessing

Before creating the feature vector for the classifier, noisy and meaningless data must be removed. We implemented the following procedure to clean up the UGC, particularly the reviews.

Step1: Remove all reviews written under the default reviewer name “A Google User”. as we cannot extract string features from the default reviewer name. The limitation introduced by this step is discussed in Section 5

Step2: Extract the reviewers who have commented on at least three apps. The limitation introduced by this step is discussed in Section 5

Step3: Remove reviews written in languages other than English.

Step4: Split all sentences into words.

Step5: Transform all letters into lowercase.

Step6: Remove all stop words such as “is”,“am”,“the”.

Step7: Consolidate variant forms of a word into a common form.(word stemming) e.g., convert “runing”to “run.”

Step8: Correct the misspelling English words for all the reviews.

From Step 3 to Step 8, we implemented natural language processing based on NLTK [9] and TextBlob [10]. 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 easily implement language detection and spelling correction in the data preprocessing stage as well as sentiment analysis during the feature extraction stage. After data preprocessing, our dataset for feature extraction included 2, 606, 791 reviewers.

2.3 Feature Extraction

We extracted a total of 15 features from the UGC and metadata. Herein, we describe how we extracted these novel features in terms of the type of data.

2.3.1 Post Time

Day Intervals PAs are likely to launch a rating promotion attack within a short day interval. According to previous work [5] reviewers hired by app promotion web services tend to complete their review promotion missions within 120 days. Given this background, we calculated the day intervals between the earliest and latest post time for each reviewer.

Entropy PAs are likely to write reviews within the same

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

day. To measure the proportion of same-day reviews, we apply the information entropy defined as follows: $H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i)$, where x_i is the frequency of same-day reviews and n is the number of apps reviewed by each reviewer. If all the reviews are posted on the same day, the entropy of the post time will be 0.

2.3.2 Review

Bi-gram Matching Posting similar reviews is also characteristic of PAs. The detection of similar reviews is more difficult than that of other corpus. One reason for this difficulty is the use of made-up words to express strong feelings, such as "goooooood" and "cooooooool." Made-up words cannot be modified by the current spelling correction algorithm, which is designed to correct misspelled words and not intentionally created words. To address this problem, we converted each word into a bi-gram bag of words and calculated the average of cosine similarity score of each pair of reviews evaluated by each reviewer. We set the threshold of cosine similarity as 0.9

Semantic similarity As mentioned in Section 1, reviewers usually use different words and expressions to express their same thoughts. To identify similar words, we applied the Paragraph Vector algorithm [6], which represents each document by a dense vector that is trained by stochastic gradient descent and back-propagation to predict the similarity of words in the document. Paragraph Vector is designed in a distributed way such that it can train a large quantity of unlabeled data in a very short time. Paragraph Vector is implemented in the Python library gensim [11]. We leveraged our large-scale dataset of 57, 868, 301 reviews to build our predictive model. Using this approach, the training of the predicted model was completed quickly, after approximately 1 hour. We considered the average of the similarity scores predicted from the trained model for each pair of reviews as our feature. Table 2 presents examples of the similarity scores computed with the trained Paragraph Vector model. It is clear that the model is able to infer 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.

Sentiment Analysis PAs tend to write positive reviews to promote apps for monetary benefit or to infect other users with malicious apps. We used TextBlob [10] to conduct a sentiment analysis of all the reviews posted by each reviewer. The sentiment analysis of 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 was used for feature vector creation. The accuracy of the sentiment analysis classifier is between 80% and 90%. In our case, both the training data and predicted data 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 used the average score for each pair of reviews predicted by the sentiment analysis classifier as our feature. Table 3 shows an example of the scores predicted by the sentiment analysis classifier.

The average length of all the reviews Fake reviews injected by PAs are likely to be short. We calculated the average length of all the reviews written by each reviewer as our feature.

2.3.3 Rating

True Reputation Algorithm As previous works [4], [5] have shown, PAs usually give high ratings to their target apps; these attacks are thus called rating promotion attacks. In official app stores, mobile users select apps using the average of all the ratings for each app as a reference. This type of metric is extremely vulnerable to rating promotion and/or demotion attacks. As a defense against these attacks, the TRUE-REPUTATION algorithm [1] calculates the true reputation score r_m of each app, which can replace the average of all the ratings as a metric. In our study, to identify a promotional attacker behavior related to rating promotion/demotion attacks, we computed our feature as $\frac{\sum_{i=1}^{n}(r_i - r_{mi})}{n}$, where n is the number of apps reviewed by each reviewer.

Average ratings PAs tend to only post reviews with high ratings. We computed the average of all the ratings posted by each reviewer as our feature. When a promotional attacker behavior occurs, the average of all ratings is close to 5.

Coefficient of variation of ratings For the same reason mentioned above, we calculated 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. If a reviewer posts identical ratings, the coefficient of variation will be 0.

2.3.4 The number of Installs

Average number of installs As the number of installs is one of the metrics when an user reviews an app, we include this metrics to characterize the feature of an app. We calculated the average number of installs for each app as a feature.

Coefficient of variation of the number of installs For the same reason mentioned above, to measure the distribution if installs, we calculated the coefficient of variation of the number of installs for each app. If a reviewer posts reviews to apps with the same number of installs, the coefficient of variation will be 0.

2.3.5 Developer and Category

Entropy It is natural that PAs are more likely to promote apps produced by the same developer because the targeted malicious apps should be associated with each other. We applied entropy as

described earlier to measure the degree to which the developer is the same. If the reviewer posts on apps with the same developer, the entropy of the developer will be 0.

2.3.6 Reviewer name

Length of reviewer name Legitimate reviewers usually choose their own name as the reviewer name, whereas the reviewer names of PAs are likely to be unusually short or long. Thus, we computed the length of the reviewer name as our feature.

Number of digits and symbols in reviewer name The reviewer names of PAs are often randomly generated by a spammer tool and are therefore likely to contain digits and symbols such as "!", "*", "@."To this end, we calculated the number of digits and symbols in the reviewer name as our feature.

2.4 Detection model

We built our detection model based on the machine-learning algorithm implemented in the Python library scikit-learn [12]. Scikit-learn is a simple and efficient tool designed for data mining and data analysis. 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, adaBoost, and GradientBoosting. 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 3. Finally, we applied the best detection model to perform a large-scale analysis of our real-world dataset.

3. Performance Evaluation

In this section, we evaluate the accuracy of PADetective system. We first present how we made the labeled dataset, i.e., the *ground truth*. We then describe the flow of evaluating the detection models. Finally, we demonstrate the accuracy of the PADetective system, using the labeled dataset.

3.1 Training set

We first generate training set, which can be used as the ground truth. We define a PA as a reviewer who posts reviews only to malicious apps. Similarly, we define a legitimate reviewer as a user who posts reviews only to benign apps. To determine whether an app is malicious or not, we utilized the outputs of VirusTotal [13]. VirusTotal is a free online service that can be applied to identify malicious files and URLs. When a user submits an app as a query, VirusTotal verifies the potential maliciousness of this app using different antivirus software and scan services. With regard to the verification of mobile apps, VirusTotal usually classifies malicious apps into two categories: malware and adware. In our study, considering that PAs most likely evaluate both malware and adware apps, we did not distinguish between these categories. With this approach and additional manual inspection, we were able to extract 723 of PAs. We also extracted legitimate users and randomly sampled 1,000 of them. The reason why we randomly sampled legitimate users is to make a good balance between the two classes when we train our classifiers.

3.2 Evaluation Method

To ensure that a selected model can identify *unknown* PAs, we randomly divided the labeled data in a 70%/30% manner. 70% of labeled data is used to optimize each machine learning model and select the best model among them. For optimizing the machine learning algorithms, we used the 10-fold cross-validation and grid search. After we select the best model, we check its performance using the rest of 30% of labeled data as test set. Since 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, we used three common metrics: false positive rate (FPR), false negative rate (FNR) and accuracy (ACC), which are computed as $FPR = \frac{FP}{FP+TN}$, $FNR = \frac{FN}{TP+FN}$, and $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, respectively, where TP is true positive, FP is false positive, TN is true negative and FN is false negative.

Table 4 Accuracy of classification on training set. Means and standard deviations are calculated from 10-times 10-fold CV 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
GradientBoosting	**0.961**	**0.016**	**0.063**	**0.028**	**0.020**	**0.035**

Table 5 Evaluation of accuracy using test set. Note that we did not use test set to train the classifier.

Machine learning Algorithm	ACC	FPR	FNR
GradientBoosting	0.954	0.080	0.011

3.3 Evalutaion Result

Table 4 presents the classification accuracy achieved using each machine learning algorithm. Most of the machine learning algorithms predicted the PAs with high accuracy and a low false negative rate, proving that our features can improve the accuracy of numerous machine-learning algorithms. Of the eight machine-learning algorithms we tested, GradientBoosting achieved the highest accuracy (96.1%) with the lowest false positive and false negative rates. The standard deviations of the accuracy, false positive rate, and false negative rate of GradientBoosting are low, indicating that GradientBoosting can identify PAs effectively and without bias. We used a grid search to determine the best parameter for GradientBoosting, finding that the optimal number of trees is 200. Based on these results, we selected GradientBoosting as our detection model.

To better understand the sources of false negative rate and false positive rate of our system, we conducted error analysis with manual inspection. We found that our system failed to identify PAs who posted reviews in a long period of time, such as two years. On the other hand, our system wrongly flagged the legitimate reviewers whose behavior was similar to a PA, e.g., their reviews seemed to be fake, but the apps reviewed were not detected by the VirusTotal. We note that using the online virus checkers

Table 6 Statistics of detected PAs 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 PAs	289,000	135,989	20,906	–
Detected PAs with high confidence	1,000	2,904	486	148

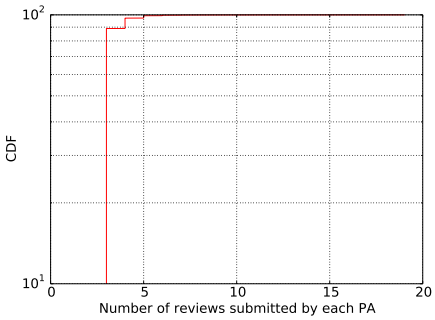

Fig. 5 Distribution of the number of apps reviewed by each PA.

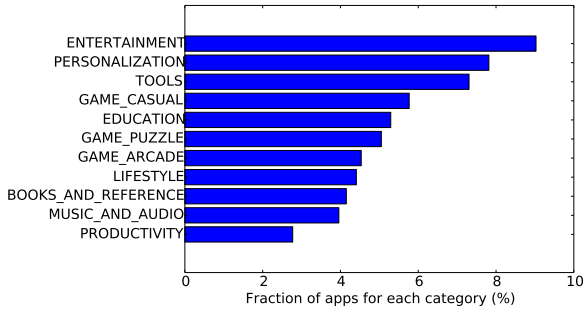

Fig. 6 Top-10 categories of apps reviewed by the detected PAs.

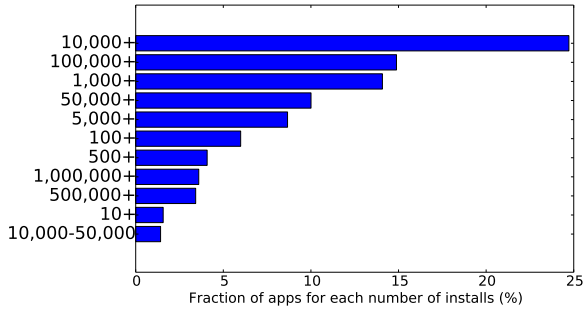

Fig. 7 Number of installs for apps reviewed by the detected PAs.

could be one of the sources of false detection.

Finally, using the optimized GradientBoosting algorithm, we test its accuracy using the test set. Table 5 shows the results. We can see that the accuracy is good for the unknown set, indicating the classification scheme is robust. In the next section, we will use this classification model to study the PAs in the large-scale data.

4. Promtional attacks in the wild

4.1 Large-scale Measurement

Using PADetective, we conducted a large-scale analysis of real-world data that had been collected from the Google Play Store. Using the labeled reviewers, i.e., 1,000 legitimate reviewers and 723 PAs, as training set, PADetective extracted 289,000 of potential PAs from the 2,605,068 of reviewers. Table 6 summarizes the number of reviewers/apps detected with PADetective. The number of unique malicious apps reviewed by the potential PAs were 20,906, accounting for approximately 65% of the malicious apps reviewed by some users. It is somewhat surprising that many of malicious apps having reviews were associated with the potential promotional attackers. Note that majority of malicious apps detected with VirusTotal had no user reviews. Next, we look at the top 1,000 reviewers detected as PAs with high confidence, which were obtained from the output of the classifier; i.e., we ranked the reviewers in descending order by the probability of being a PA. 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. Of the 1,000 PAs, 136 reviewers (13.6%) posted reviews only for malicious apps or the deleted apps. We found that other detected reviewers posted reviews not only for malicious apps, but also for apps that were not detected by the VirusTotal. We leave checking the code of those undetected apps for our future work.

Figure 5 plots the cumulative distribution (CDF) of number of reviews submitted by each potential PA. While there are few potential PAs that submitted reviews for many apps, majority of the potential PAs submitted reviews on three apps, which was the threshold we set as a minimum number of apps reviewed by each PA. We conjecture that this fact reflects the fact that PAs do not submit many reviews, using a same account name so that bogus reviews are not detected.

Next, we investigated the categories of apps reviewed by PAs. Figure 6 presents the results. Three out of ten categories (approximately 15% in total) are related to game, which was the primary target of the promotion attacks. We also observe that the categories of targeted apps span across various categories. To study the impact of apps promoted by the PAs, we analyzed the number of installs for apps reviewed by the detected PAs. Figure 7 presents the results. We see majority of of apps reviewed by the PAs had rather less number of installs. This observation indicates that PAs tend to target apps with moderate popularity.

Finally, we investigate whether detecting PAs can be used to discover malicious apps. Namely, we compared the time at which the PAs posted reviews on malicious apps and the time when the malicious app was first submitted to VirusTotal. If all the post times are earlier than the first submission times, then our PA detection scheme has the potential to identify new, previously unknown malicious apps soon after publication. For this study, We used the top 241 of detected PAs who reviewed only for malicious apps. We found that of the 241 of detected PAs, 72 of them reviewed malicious apps before the malicious apps were checked by the VirusTotal. Of the all apps reviewed by these 72 PAs, 217

Table 7 A set of apps reviewed by a detected PA.

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 8 A set of apps reviewed by a detected PA.

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.billinfo.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

of apps were detected as malicious by the VirusTotal. We also note that other apps reviewed by the PAs can be used to pick up suspicious apps that need to be inspected.

In summary, the PADetective system detected 289 K of reviewers as potential PAs. The detected potential PAs posted reviews to 136 K of apps, which included 21 K of malicious apps. Among the top-1K 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 where more than half of apps were detected as malicious. Our analysis also found that PADetective can be used to detect malicious apps early in the app publication process.

4.2 Case Study

Herein, we present two examples of PAs as a case study to demonstrate the effectiveness of our PADetective system.

Promotional attackers in the wild Table 7 presents the example of a PA detected by our PAdetective system. This PA gave high ratings and posted similar positive reviews to 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 difference between the true reputation scores and ratings was larger than 1.0, which indicates the reviewer attempted to promote all the malicious apps using high ratings. This finding clearly illustrates our assumption that the PA really exists in the wild. By detecting PAs at an early stage, we have chances to discover malicious activities on the app market.

Detection of previously unknown malicious apps As shown in table 8, this reviewer gave high ratings and wrote very short positive reviews for three malicious apps on the same day. Moreover, all the post times are earlier than the first submission times in VirusTotal. The apps evaluated by this promotional attacker are 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.

5. discussion

In this section, we discuss some of the limitations of PADetective and future research directions derived from these limitations.

Reviewer integrity In this study, we used the reviewer name as each reviewer's identifier. An attacker may inject fake ratings and reviews for different malicious/benign apps using different reviewer names, as it is easy and free to obtain numerous reviewer names. PADetective will fail to detect promotional attackers who use a variety of reviewer names to post fake ratings and reviews for only one or two apps. In this case, additional features collected by app store providers, such as IP addresses or geographic information, can be added to PADetective so that PADetective is able to identify such reviewers as PAs.

Number of apps reviewed by each reviewer As mentioned in the data preprocessing step, we removed reviewers who review only one or two apps because the number of apps reviewed by each reviewer is too small to allow the extraction of enough features. The number of apps reviewed by each reviewer will probably reach or exceed 3 over time, at which point these reviewers can be inputted to PADetective. In future study, we will build a real-time collection system that can refresh the UGC and the metadata within a very short period, such as one or two days, instead of an offline collection system.

Maximum number of reviews We only crawled 4,500 most recent reviews due to the constraint of Google Play review collection API. This may lead to false negative that our system miss to discover the PAs who exhibit their malicious behavior at very early time such as five year ago. We aim to identify the newest promotional attackers who can be used to find our potentially malicious apps before they were submitted to public online virus checkers. To keep our collection data up to date, we need to build a real-time collection system mentioned above.

6. Related work

Numerous analysis methods related to our system have been proposed in recent years. Such methods can be classified into three categories depending on what type of data is used. In this section, we review related work from these three categories and compare them with ours.

UGC analysis Kong et al. [14] designed a system called AutoREB, which can automatically understand users ' mobile app security and privacy concerns. They applied the relevance feedback technique for the semantic analysis of user reviews and the crowdsourcing technique to aggregate the results of user review analysis to apps' behaviors. Mukherjee et al. [2], [3] studied the detection of fake reviewer groups from Amazon product reviews. They first used a frequent itemset mining method to identify a set of candidate groups and then used several behavioral models based on the relationships among groups such as the review posting times and similarities. Xie et al. [4] presented a method that can discover collusion reviewer groups in app stores. They built a relation graph using rating and deviation of ratings and then applied graph cluster algorithm to detect collusion reviewer groups. Our PADetective system can make use of all UGC, including post time and reviewer name, to identify previously unknown promotional attackers effectively and efficiently. Moreover, we performed a large-scale analysis on real-world data collected from approximately 1 M apps.

Metadata Analysis Xie et al. [5] studied the mobile app reviews traded on the underground market. They analyzed the metadata of promoted apps collected from the underground market, including average ratings, total number of reviewers, category distributions, and developers. The WHYPER system [15] was the first work to analyze text descriptions semantically to perform risk assessments of mobile apps. The experimental results showed that WHYPER can accurately relate text descriptions with app permissions. Qu et al. [16] developed the AutoCog system for measuring description-to-permission fidelity in Android applications. AutoCog is most relevant to WHYPER. The advantage of AutoCog is not only its detection performance but also its ability to generalize over permissions to a large extent. Our PADetective system is able to extract features from not only the metadata mentioned above but also UGC in a more comprehensive way.

App analysis Kirin [17] 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. DREBIN [18] is a lightweight and automatic Android malware detection system that is deployed directly on the smartphone. DroidMiner [19] can automatically mine malicious program logic from known Android malware using behavioral graph and machine-learning techniques. In our study, we attempt to address this problem in a different way that utilizes the correlation between UGC, metadata, and malicious apps to improve the quality of mobile app store service. We believe that our system will be an important complement to the past research outcomes on the analysis/detection of malicious apps.

7. Conclusion

In this study, we have proposed the PADetective system, which can identify unknown promotional attackers in app stores using UGC and metadata, as well as machine-learning techniques. We applied PADetective to a large-scale analysis of unlabeled reviewer data. The PADetective system detected 289 K of reviewers as potential PAs. The detected potential PAs posted reviews to 136 K of apps, which included 21 K of malicious apps. The large-scale evaluation and case study analysis illustrated that PADetective is able to effectively and efficiently predict previously unknown malicious apps. In future study, we plan to make the PADetective system work in a real-time manner.

References

[1] H. Oh, S. Kim, S. Park, and M. Zhou, "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.

[2] A. Mukherjee, B. Liu, J. Wang, N. S. Glance, and N. Jindal, "Detecting group review spam," in *Proceedings of the 20th International Conference on World Wide Web, WWW'11*.

[3] A. Mukherjee, B. Liu, and N. S. Glance, "Spotting fake reviewer groups in consumer reviews," in *Proceedings of the 21st World Wide Web Conference , WWW'12*.

[4] Z. Xie and S. Zhu, "Grouptie: toward hidden collusion group discovery in app stores," in *Proceedings of the 7th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec'14*.

[5] Z. Xie and S. Zhu, "Appwatcher: unveiling the underground market of trading mobile app reviews," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec'15*.

[6] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31th International Conference on Machine Learning, ICML'14*.

[7] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '14*.

[8] "Google play reviews collection service." `https://play.google.com/store/getreviews`.

[9] "Natural language toolkit." `http://www.nltk.org`.

[10] "Textblob: Simplified text processing." `http://textblob.readthedocs.io/en/dev/`.

[11] "gensim:topic modelling for humans." `https://radimrehurek.com/gensim/`.

[12] "scikit-learn:machine learning in python." `http://scikit-learn.org/stable/`.

[13] "Virustotal- free online virus, malware and url scanner." `https://www.virustotal.com`.

[14] D. Kong, L. Cen, and H. Jin, "AUTOREB: automatically understanding the review-to-behavior fidelity in android applications," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security, CCS'15*.

[15] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: towards automating risk assessment of mobile applications," in *Proceedings of the 22th USENIX Security Symposium, USENIX'13*.

[16] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *Proceedings of the ACM Conference on Computer and Communications Security, CCS'14*.

[17] W. Enck, M. Ongtang, and P. D. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the ACM Conference on Computer and Communications Security, CCS'09*.

[18] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS'14*.

[19] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. A. Porras, "Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *Proceedings of the 19th European Symposium on Research in Computer Security, ESORICS'14*.